Efficient Adaptive Multi-Granularity Service Composition

Barakat, Lina

Awarding institution:
King's College London

Efficient Adaptive Multi-Granularity Service Composition

by

Lina Barakat

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the
Department of Informatics
the
School of Natural & Mathematical Sciences

November 2013

Abstract

Despite the tremendous benefits of the dynamic, service-oriented approach to build composite applications, it also brings great challenges. In particular, the run-time selection of the most suitable services for an application in a timely manner is not trivial, since many providers could be competing for the same type of service, but at different quality of service levels. Due to possible dependencies among services, such quality offerings could also vary at a single service level. This is further complicated by the fact that the available services may offer to achieve the tasks required at varying functional abstractions. Moreover, services are highly dynamic and unreliable in nature, which can cause serious problems to the execution of workflows relying on such services. In this thesis, we contribute towards addressing these challenges, and achieve a more efficient, robust, and optimal dynamic composition process.

Specifically, through a rich collection of alternative planning options, we allow services at various granularity levels to be incorporated into the selection process. We also enrich the quality model of services with inter-service dependency awareness, to produce correct quality estimations. Furthermore, we develop efficiency-boosting techniques facilitating a scalable service selection process without affecting optimality, even in the case where the search space experiences complex dependencies among services. In the face of environment dynamism and uncertainty, we achieve an *early* and *efficient* adaptive behaviour, which ensures a valid, optimal, and satisfactory solution, in spite of high environment volatility, and without causing disruption to application execution.

The effectiveness of all the algorithms and techniques developed in this thesis is demonstrated analytically and empirically. The latter is achieved both on randomly generated datasets, and through a case study evaluation applied in the context of learning object composition.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisors Professor Michael Luck and Dr. Simon Miles. I have been privileged to have two supervisors who cared so much about my work, challenged me to deliver my best, and constantly boosted my confidence and motivation through encouragement and praise. Without their invaluable support and guidance, and the tremendous time they invested (often sacrificing weekends and holidays), this thesis would not have been possible. I must also thank Dr. Iman Poernomo for his insightful comments during the annual progress review meetings.

I would also like to acknowledge with much appreciation the crucial role of Damascus University, providing me with the basic knowledge to pursue a PhD degree in the first place, and supporting me financially throughout my studies.

Additionally, I owe special thanks to my friends, colleagues, and all the members of the Department of Informatics at King's College London, who created a stimulating and very friendly working environment, making my experience incredibly valuable and enjoyable. For their endless love, support, and faith in me, I am forever grateful to my beloved family, my father Wafik, my mother Olga, my sisters Linda and Lida, and the sweetest nephew and niece, Sasha and Anna. I also extend my sincere thanks to my inlaws, Ali, Naelah, Stewart, Lama, Ghiath, Hyam, Aous, Shaam, Mufid, and the little ones Majed and Karam.

Last but not least, my love and appreciation go to my husband, Samhar, for sharing with me every difficult moment, giving me the strength to keep going, and surrounding me with the warmest love and support.

Contents

List of Figures

List of Tables

List of Algorithms

List of Notation

Sets

ADD	Services to be added to task's non-dominated services
AR	Constrained attributes
FD	Functionality descriptions
G	*Dag*s formed from tasks
G_s	*Dag*s formed from services
$NAME_a$	Quality attribute names
NTR	Nodes to be revisited
RMV	Services to be removed from task's non-dominated services
S	Available services (service space)
$SPLAN$	Plan-based pruning plans
T	Hierarchy tasks
$VALUE_a$	Quality attribute values

Functions

$aggr$	Quality aggregation function
$atindex$	Service appearing at an index
$candidates$	Task's candidate services
$comp$	Task's actual plans
$const_r$	Request's quality constraints
dir_a	Attribute direction
dm	Domination relation
dom_a	Attribute domain
$dominated_s$	Services dominated by a service

$dominating_s$	Services dominating a service
$enode$	Graph's end node(s)
$eservice$	Instance's last service
$func_s$	Service functionality
$func_t$	Task's functionality
$graph_t$	Task's decomposition graphs
$ideal_t$	Task's ideal quality values
$indexof$	Node's index in a path
$instances$	Task graph's instances
ir	1− instance pruning rate
$match$	Matching function
max_r	Request-level max quality values
max_t	Task's max quality values
min_r	Request-level min quality values
min_t	Task's min quality values
$name_s$	Service's attribute names
$nodes$	Graph nodes
$oinstances$	Task's optimal instances
$plan$	Task's abstract plans
$plnnum$	Number of abstract plans
$rcandidates$	Task's non-dominated services
$r - dm$	Request-based domination relation
$satisfies$	Satisfaction status of a condition by a composite service
$scaled_c$	Composite service's scaled quality values
$scaled_s$	Service's scaled quality values
$snode$	Graph's start node(s)
sr	1− service pruning rate
$status$	Valid predecessor's status
$task_r$	Requested task
$utility_c$	Composite service utility
$utility_s$	Service utility
$validpred$	Task's valid predecessors
$value_c$	Composite service's quality values

$value_s$	Service's quality values
$weight_r$	Request's quality weights
τ	Time complexity

Chapter 1

Introduction

1.1 Introduction

Service-oriented computing (SOC) is becoming the dominant paradigm for low-cost, time-effective application development, permitting a flexible compositional approach to building complex distributed applications via the integration of loosely-coupled software components, offered as services by external providers. Such service-based applications are increasing in scale, with a vast range of resources and functionalities currently being exposed as services over open networks (e.g., the Web and Grid systems), and it is already beyond human ability to analyse and compose these services manually. Hence, enabling the automated discovery and composition of services in such open distributed systems has recently emerged as a major research topic. Yet, despite active research, current service composition approaches have still not dealt adequately with the huge amount of available services that differ in their granularities (amount of functionality provided) and offered qualities (e.g. price, execution time, reliability), the correlations possibly existing among these services, and the dynamic nature of the environment

(services can enter or leave the system at any time, or change their characteristics). Addressing these challenges of service composition is the main focus of this thesis.

The rest of the chapter is organised as follows. The service-oriented computing paradigm is introduced in Section 1.2. Section 1.3 discusses the dynamic composition of services (the specific area of interest), and highlights the challenges and limitations in existing work that the thesis aims to address. In light of the challenges identified, our research aims and contributions are listed in Section 1.4, before concluding in Section 1.5 with an outline of the reminder of the thesis.

1.2 Service-oriented Computing

The advances in network and communication technologies have made it possible for independent computer systems residing at different sites to interact with each other over networks. Such connectivity brings many advantages to various parties including organisations, businesses, societies and individuals. For example, individual users connected to open networks, such as the Internet, have access today to a vast amount of information, goods and services [128]. Likewise, scientific communities across the world can benefit from sharing high-specification computing power, storage facilities, as well as large data repositories via high-speed networks [50]. The business world is also a major beneficiary since enterprises can utilise such distributed communication capabilities not only to advertise and sell their products and services to end-users in an efficient and cost-effective manner, but also to automate their interactions with their trading partners (e.g. suppliers), which results in increased productivity and flexibility in business process development, and the ability to pursue new market opportunities [61].

Service-oriented computing (SOC) has emerged as a suitable paradigm for realising distributed applications of such a heterogeneous nature [98], i.e., where the interacting

software components are possibly developed by independent providers, implemented in different languages, and scattered across distinct platforms. In this paradigm, providers encapsulate their offerings, which could range from hardware to whole applications [102], within *services* and expose them through well-defined interfaces on a network of customers (e.g., the World Wide Web [26]). Services are self-contained, platform-independent computational entities that are described, published, discovered and invoked over the network using accepted standard languages and protocols [99]. Through their accessibility, reusability, and loose coupling, services provide the building blocks for the quick and cost-effective development of flexible large-scale distributed applications that may span organisational and enterprise boundaries.

Web services technology is a commonly adopted (but not the only possible) implementation technology of SOC services. It utilises the Internet as the communication medium, and relies on XML-based standards to achieve interoperability among heterogeneous communicating parties. Specifically, web service interfaces are described using a special XML-based language called WSDL (Web Services Description Language) [36]. These WSDL-based service descriptions are made available through UDDI (Universal Description, Discovery and Integration) service registries [24], in order to enable their discovery by potential clients. The invocation of web services by client applications is achieved through SOAP (Simple Object Access Protocol) [53], an XML-based protocol for sending messages over HTTP.

A key feature enabled by service-oriented computing is the dynamic binding mechanism. That is, instead of having the services hard-coded into the application at design time, which requires expensive and time-consuming maintenance to accommodate any change [124], services can be discovered and integrated on the fly at run time. Such late-binding of services brings great flexibility and agility to distributed systems development, especially against the dynamism and uncertainty usually inherent in such

systems. Among its many benefits are *increased availability* (providers are selected when needed based on the currently available market offerings, without the need to rely on the availability of any particular provider pre-selected at design time), *personalisation and customisation* (services can be flexibly selected to meet and optimise a vast range of varying criteria, for instance due to the different needs of individuals or changes in the market demand, which are simply not possible to anticipate at design time), and *rapid adaptivity and fault tolerance* (services can be easily and transparently substituted with alternatives in case of an error or an emerging new opportunity).

Despite its benefits, such dynamic integration of services at run time poses new complications and challenges that are still not addressed adequately by current research. We highlight these challenges in the next section.

1.3 Dynamic Service Composition: Research Scope and Challenges

Service composition is at the heart of SOC, allowing combination of the capabilities of multiple network-accessible services in order to achieve some complex task that cannot be met by any individual service [66]. In fact, the greatest value of having service-based systems lies in such an ability to compose new applications through reusing existing services [61], thus saving significant time and cost while delivering better services and fulfilling a wide range of needs. For example, these compositional advantages are out of the main reasons behind the emergence of business web services and virtual supply-chains [37]: networks of collaborating business partners working together and integrating their activities to gain competitive advantages in today's demanding and volatile markets in which it is difficult for individual organisations to survive alone.

Commonly, *workflows* are the paradigm adopted for representing and managing composite activities in distributed applications [140] [51], specifying the collection of tasks needed to achieve a particular high-level goal, along with coordination of their data and control. These tasks are performed by services scattered across the network and (possibly) provided by different autonomous parties. Due to the advantages stated earlier, there is growing interest in shifting towards *dynamic* binding of services [34, 46, 147, 67]. That is, the workflow is defined in an abstract form, with its tasks being bound to actual services at run time based on specific needs and service availability, as opposed to *static* binding where a concrete service instance is fixed for each task at design time.

Intuitively, the dynamic approach demands the ability to *automatically discover* and *invoke* services by a software application at run time. During the past years, a large body of research has been concerned with the automation of these operations. Industry has already developed a number of standards and protocols that facilitate locating and invoking services in heterogeneous environments with minimal human intervention, while the semantic web community has contributed towards providing expressive, machine-understandable descriptions of services, and effective matchmaking techniques to measure the suitability of a service against a required capability [87, 117]. In this thesis, we thus assume that automatic discovery and invocation mechanisms are in place, and focus instead on the problem of automatically *selecting* and *maintaining* the best combination of services at run time for a particular complex goal.

To differentiate between functionally equivalent services, and to accommodate the different expectations of users (individuals or organisations), the selection of services for the workflow tasks, besides functional suitability, should also be guided by non-functional (quality of service) properties. Examples of such properties include price, reliability and response time. Current quality-based dynamic selection approaches are

far from being satisfactory, especially when dealing with the characteristics of large-scale open distributed systems, where the interacting services are distributed among various locations, and are beyond the control of customers or any single organisation. A number of challenges associated with such systems, which this thesis aims to address, are discussed next.

1.3.1 Granular Heterogeneity

With the absence of any standard or uniform methodology indicating the *right* functional granularity for a service [56], and the autonomous nature of the providers participating in distributed systems, services are available at a vast variety of granularities. Yet, the current practice of structuring an application as a graph of tasks of specific granularity levels may prevent the discovery of many *good* services not meeting these particular granularity constraints. For example, a provider offering storage and delivery facilities as one service will not be considered if storage and delivery are modelled as separate tasks in a workflow. Alternatively, the combination of these two tasks in a workflow will lead to neglecting individual storage and delivery services. Thus, in order to utilise the available service landscape more effectively, there is a need for a richer representation of composite applications.

1.3.2 Time Constraints

A critical feature imposed by the dynamic binding of services is the sensitivity towards time limits. Since services are selected at run time, it is crucial that this selection is completed within a reasonable time frame; otherwise, it simply becomes impractical. In the case of a single-task application, or a closed system with only a limited number of services, selecting services in a timely manner is easy. However, achieving scalable

dynamic selection becomes very challenging in large-scale distributed systems. Two factors contribute in this regard.

First, such systems are normally multi-tasked, and engage a large number of providers offering identical functionalities, but different non-functional properties. For instance, today's virtual enterprises are faced with an enormous amount of competing partners to choose from when instantiating business processes. Computational grids are another example of large-scale systems, providing a vast pool of remote resources (such as processing power and data storage), each possibly optimising different quality measures, to serve the on-demand execution of complex scientific workflows.

Another factor is that, with ever-demanding users requiring high-quality services at low cost, organisations and businesses are pressured to deliver highly personalised solutions that meet user needs in an *optimal* way (e.g., in order to survive intense competition in the market). In many cases, user requirements span multiple tasks, specifying non-functional (quality of service) criteria at the workflow level (e.g., total price and total time). Such global (end-to-end) constraints necessitate the exploration of all available service combinations, since individually selecting the best service for each task, without considering other tasks, is not sufficient to guarantee their satisfaction.

The above two factors taken together result in service selection suffering combinatorial explosion. Despite some current efforts to overcome this problem, these either do not scale well, or compromise solution optimality in favour of achieving selection efficiency.

1.3.3 Dynamism and Uncertainty

Open distributed service-based systems usually exhibit high degrees of dynamism and uncertainty, as a result of which services may not behave as expected or might not be available when needed. This can be due to several reasons, either intentional or

unintentional. For example, service providers, being autonomous and self-interested, may choose to act maliciously and announce false capabilities in order to increase their own profit by attracting more customers. Even in cases where the providers are fully cooperative, it might be difficult (or simply not possible) to guarantee the availability or specific quality values for a service, because of their dependency on various run-time factors. For instance, the service response time at any particular moment could largely be affected by the provider load and network traffic at that moment. Similarly, the service could suddenly become unavailable due to network or hardware failure. Moreover, given the open nature of the environment, new providers with better offerings could join the system, while existing providers could leave or change their service quality features (e.g., driven by competition) at any time.

Although the dynamic binding of services offers some tolerance against such dynamism and uncertainty, it does not guarantee the successful execution of the workflow, i.e. that the services are available on invocation and deliver what is anticipated or promised at the time of selection. This is because the selection step normally takes place prior to the start of execution. That is, services are first selected for *all* the tasks in order to be able to reason effectively about workflow-level non-functional properties. Hence, changes to a selected service could occur at any time before the actual invocation of this service, especially when executing complex workflows involving many tasks (the case with most realistic applications). Furthermore, better services could emerge during this pre-invocation period, making the selected combination of services no longer the best option. Most state-of-the-art approaches fail to address these issues without causing long disruptions in workflow execution or considerably degrading the quality of the composite application, thus necessitating better change handling mechanisms.

1.3.4 Service Correlations

In open distributed systems, services span multiple geographical locations, apply heterogeneous implementation technologies and are owned by self-interested parties. Given these features, it is not unlikely for the same service to behave differently, with respect to non-functional properties, in different composition scenarios. A possible reason for such quality diversity per service could be the business correlations very often encountered among enterprises. For example, complementary providers may arrange to offer discount policies applicable only in the case where their services are selected together. Likewise, they may choose to harm a competitor by announcing lower quality levels if integrated with the latter's services. Other factors contributing to the quality variations of the same service could include methodological compatibilities, e.g., a software testing service achieves a better accuracy when combined with a particular software generation service, or network distance, e.g., combining a resource with its local counterparts (as opposed to the remote options) improves its response time due to the elimination of network latency factors.

Inter-service quality correlations are largely ignored in the service composition literature. Instead, services are usually assumed to be independent in this respect, and deliver identical quality levels regardless of the composition context. Such an assumption could greatly affect the ability of the selection algorithm to produce accurate quality estimates for service combinations, thus risking the selection of non-optimal combinations, while not considering promising ones. Based on this, it is vital for the selection algorithm to account for quality correlations among services, but again special attention should be paid to the scalability problem, especially with the further complexity introduced by these correlations.

1.4 Research Aims and Contributions

This thesis aims to advance the state of the art in the area of quality-aware dynamic service selection and execution, by presenting suitable techniques and algorithms capable of handling the challenges and limitations identified in the previous section. Although the target application areas are large-scale open service-based systems (e.g., large-scale virtual enterprises and Grid systems), a simple travel planning scenario will be used throughout the thesis for illustration purposes.

The contributions of the thesis are summarised below.

First, to cope with the diversity in service granularity levels, a hierarchical representation of composite applications, with multiple decompositions of tasks, is proposed, allowing the discovery of relevant services at arbitrary levels of granularity.

Second, to improve selection efficiency, user-specific search space reduction techniques are introduced, the application of which, prior to performing service selection, results in significant improvement in selection performance in terms of execution time, without affecting solution optimality. The first and second contributions have been published as [23]:

L. Barakat, S. Miles, I. Poernomo, and M. Luck. Efficient multi-granularity service composition. In *Proceedings of the 2011 IEEE 18th International Conference on Web Services*, pages 227–234, 2011

Third, a novel reactive selection algorithm is presented, capable of adapting efficiently to service changes during service selection, thus producing the best possible composition for the user in the current environment state. That is, the selected solution is executable, satisfactory, and optimal prior to execution. This work has been published as [22]:

L. Barakat, S. Miles, and M. Luck. Reactive service selection in dynamic service environments. In *Proceedings of the First European Conference on Service-Oriented and Cloud Computing*, pages 17–31, 2012

Fourth, the reactive selection algorithm is extended to efficiently handle service changes occurring at execution time, for both repair and optimisation purposes. The adaptation is performed as soon as possible and in parallel with the execution process, thus reducing interruption time, increasing the chance of a successful recovery, and producing the most optimal solution according to the current environment state.

Finally, a correlation-aware composition approach is introduced, where quality dependencies among services are modelled and considered during composite service selection, thus producing more accurate quality estimations of service combinations and providing users with better solutions. Moreover, to improve selection efficiency, correlation-aware search space reduction techniques are presented, which prune out uninteresting service compositions prior to selection while accounting for the correlations among services. This work has been published as [21]:

L. Barakat, S. Miles, and M. Luck. Efficient correlation-aware service selection. In *Proceedings of the 2012 IEEE 19th International Conference on Web Services*, pages 1–8, 2012

1.5 Thesis Outline

The rest of the thesis is organised as follows. Chapter 2 reviews related literature on service composition, focusing particularly on quality-based dynamic approaches. A multi-granularity planning knowledge representation, comprising tasks at different hierarchical levels, is introduced in Chapter 3. Based on this, the various components

involved in a quality-based service selection problem are modelled formally. In Chapter 4, pruning techniques are developed to reduce the size of the search space prior to selection. A service selection algorithm, incorporating these techniques, is also presented in the same chapter, followed by theoretical and empirical evaluation to assess its effectiveness. A reactive version of the selection algorithm, which adapts efficiently to environment dynamism while performing selection, is outlined in Chapter 5. Again, the algorithm is evaluated both theoretically and empirically. This is followed in Chapter 6 with an extension to address execution-time service changes. A categorisation of changes based on their importance and urgency is also provided, along with the corresponding behaviour of the executing system. Chapter 7 accounts for quality correlations among services, and presents a correlation-aware quality model of services, a correlation-aware selection algorithm, and correlation-aware search space reduction techniques that reduce the time complexity of selection. As before, a theoretical and empirical evaluation are performed to prove effectiveness. To support the experimental results obtained throughout the thesis using randomly-generated data, a case study evaluation, applied on the realistic domain of learning object composition, is detailed in Chapter 8. The domain is first introduced, and then mapped to the quality-based service selection problem of interest in this thesis. A thorough empirical evaluation is then conducted for all developed algorithms. Finally, Chapter 9 concludes the thesis with a research summary and a list of future directions.

Chapter 2

Literature Review

2.1 Introduction

Having set the research goals in Chapter 1, we provide in this chapter a review of state-of-the-art research in the area of dynamic service composition, focusing mainly on current efforts towards quality-of-service awareness and adaptive behaviour during service selection and execution, to which we aim to contribute. We start with a brief overview of service-oriented computing in Section 2.2, which adds to our previous introduction. In Section 2.3, we present current trends in dynamic service composition, which can be broadly categorised into AI-planning-based and workflow-based approaches, before turning our attention to the latter (the context of this thesis) in the rest of the chapter. Specifically, the problem of selecting suitable services for workflow tasks, while accounting for the non-functional properties of these services, is discussed in Section 2.4, summarising the main approaches in this regard. This is followed, in Section 2.5, by a detailed review of current solutions to achieving robust workflow execution in dynamic and uncertain service environments. Some of these solutions take preventive actions to

avoid the occurrence of run-time exceptions, while others concentrate on identifying appropriate exception handling mechanisms for effective fault-tolerance. Existing limited efforts on inter-service quality-correlation awareness during workflow instantiation are outlined in Section 2.6, before concluding in Section 2.7 with an analytical summary.

2.2 Service-oriented Computing

Service-oriented computing (SOC) is a promising paradigm for the sharing of resources and functionalities in large-scale, open, distributed environments (e.g., the web and computational Grids). Via exposing such resources and functionalities as *services* (self-describing and platform-independent reusable software entities), and utilisng these services as elementary building blocks, this paradigm supports the rapid and economic development of complex, interoperable distributed applications. The basic service-oriented architecture (SOA) [52], adopted in SOC, is depicted in Figure 2.1. It involves three roles, the service provider, service consumer, and service registry, interacting according to three operations, publish, find, and bind. Specifically, a service provider describes its service using a standard format, and publishes this description in a public service registry so that the service can be discovered by potential clients. A service consumer, which could be an end-user application or another service, searches the service registry to find a required service, and retrieve its binding information. The binding details obtained from the service description can then be utilised by the consumer to locate and invoke the service. For example, in the context of web services (the primarily used technology for realising SOC), services are described by WSDL (Web Services Description Language) documents, which define service interfaces and invocation mechanisms. WSDL descriptions can be made accessible to clients by advertising them in a UDDI (Universal Description, Discovery, and Integration) service registry,

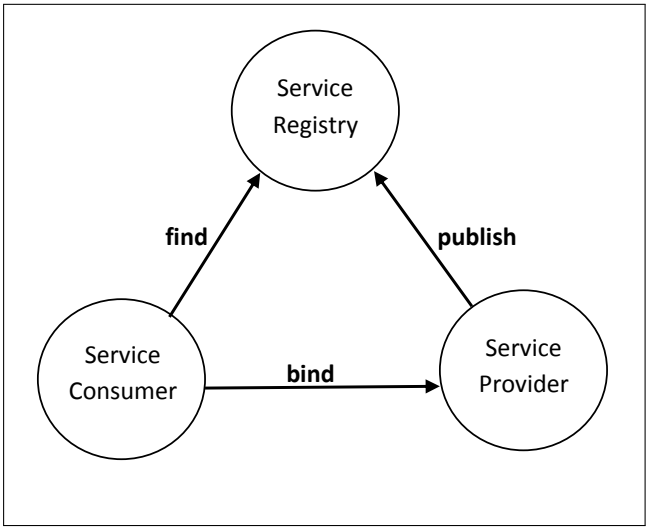

FIGURE 2.1: Roles and interactions involved in the basic SOA

while the interactions between consumers and providers concerning service invocations are achieved through SOAP (Simple Object Access Protocol) messages.

Standard languages, such as WSDL, participate towards addressing the interoperability issue among the heterogeneous interacting parties in service-based systems, via providing a standardised way for describing services. Yet, such description is purely syntactic and lacks an explicit semantics, thus largely restricting the ability to reason *dynamically* about the suitability of services for a requirement at hand. Consequently, to facilitate the dynamic discovery of services, many efforts have been concerned with providing machine-understandable, *semantically-enriched* descriptions for services, and corresponding semantic matching capabilities. Proposals supporting the semantic markup of services include: OWL-S [82] (formerly DAML-S [11]), a web service ontology based on OWL (the Web Ontology Language); WSMO (the Web Service Modeling Ontology) [105, 134]; and the W3C recommendation SAWSDL (Semantic Annotations for WSDL and XML Schema) [72]. Based on these formalisms, several semantic matchmakers have been proposed [117, 113, 68, 70, 71, 93], which allow inference of logical equivalence between services and requests by reasoning over their

semantic descriptions, thus effectively identifying appropriate matches against a particular request, even with the presence of syntactic differences in content representation, typical in the heterogeneous, inter-organisational settings of service-based systems. It is worth noting here that enabling the dynamic discovery of services is the first step to enabling their dynamic composition. In this thesis, however, we will not address this step directly, but instead assume that it is in place (or at least under ongoing, separate research).

Service descriptions could also reference the quality of service (QoS) characteristics of services, indicating their non-functional capabilities. These attributes can be generic, such as price, response time, and reliability, or domain-dependent, representing specific features and metrics of a particular domain. To support the modelling of quality-related aspects, extensions to various web service description languages, for example, have been proposed, including WSDL [45], OWL-S [73], and WSMO [122]. Generally, the QoS information of a service can be directly published into public service registries by the service provider [103], assessed from monitoring previous service performance by specialised proxies [85], or negotiated with the provider in terms of service level agreements (SLAs) [41, 10]. In this thesis, we do not focus on any particular acquisition method, and simply assume that access to QoS information of services is available.

A fundamental advantage of the above service technologies is providing support towards enabling *automated* service composition (a major reason behind their emergence), which is discussed in the next section.

2.3 Automatic Service Composition

Very often, users want to achieve high-level goals that cannot be fulfilled by any individual service. Such goals require aggregating the functionalities of multiple existing

services into more sophisticated composite applications. Since SOAs are increasing in scale, it is very difficult, if not almost impossible, to analyse and compose the available services manually by a human user. Hence, automatic service composition has gained much attention in recent years, and several approaches have been proposed, which can mainly be categorised into [104]: *AI-planning-based* and *workflow-based* approaches.

2.3.1 AI Planning based Composition

In the AI planning-based category, the service composition problem is viewed as a planning problem, and thus uses AI planning algorithms to combine services automatically. The main idea of such approaches [111, 125, 69] can be summarised as follows: given the requester's goal, and the list of available services described in terms of their inputs, outputs, preconditions, and effects using a semantic description language (OWL-S, for example), the planner attempts to find sequences of services that satisfy the requested objective. Specifically, while the planner views the user's objective as the goal state to be reached, services are considered as possible actions that can alter the state of the world (executing a service results in changing its precondition(s) state to its effect(s) state).

Although AI-planning-based methods automate the entire service composition process, they have a high computational cost, and thus cannot be performed in realistic time in environments with large numbers of services, especially in the common cases where quality-based reasoning is required. Therefore, in the rest of this thesis, we turn our focus to the more scalable, workflow-based approaches.

2.3.2 Workflow based Composition

The workflow-based category depends on manually-created *abstract* workflows to guide the composition process. Such workflows can be viewed as abstract plans for achieving complex goals, specifying the abstract tasks to be fulfilled along with their ordering conditions (data and control flow), but without referring to actual services. Specifically, each abstract task in the workflow is associated with search criteria describing the appropriate services for this task [110], which can then be selected dynamically at run time based on service availability and QoS (non-functional) requirements (such QoS-aware service selection is discussed in Section 2.4).

The tasks of a workflow can be connected according to multiple structures, of which, the most common are sequential, parallel, exclusive choice, and loop structures [1, 64, 143, 32]. The *sequential* structure represents a consecutive execution of tasks. For example, in Figure 2.2(a), task t_j is enabled after task t_i is completed. The *parallel* structure represents concurrent execution of tasks. For example, in Figure 2.2(b), tasks t_i and t_j can either be executed simultaneously, or in any order. This structure normally involves the *AND-split* and *AND-join* flow patterns, which divide the control into multiple parallel threads, and synchronise these parallel threads into a single thread, respectively. The *exclusive choice* structure represents an alternative execution of tasks. That is, in Figure 2.2(c), either task t_i or task t_j should be selected for execution (but not both). This structure normally involves the *XOR-split* and *XOR-join* flow patterns, which divide the control into alternative branches, and merge these branches, respectively. Finally, the *loop* structure represents a repeated execution of a particular construct (Figure 2.2(d)). A more comprehensive classification, including other structures, is provided by van der Aalst et al. [1].

Various specification languages have been proposed for modeling and controlling the execution of workflows. These languages can be classified into [137, 60]: graph-based

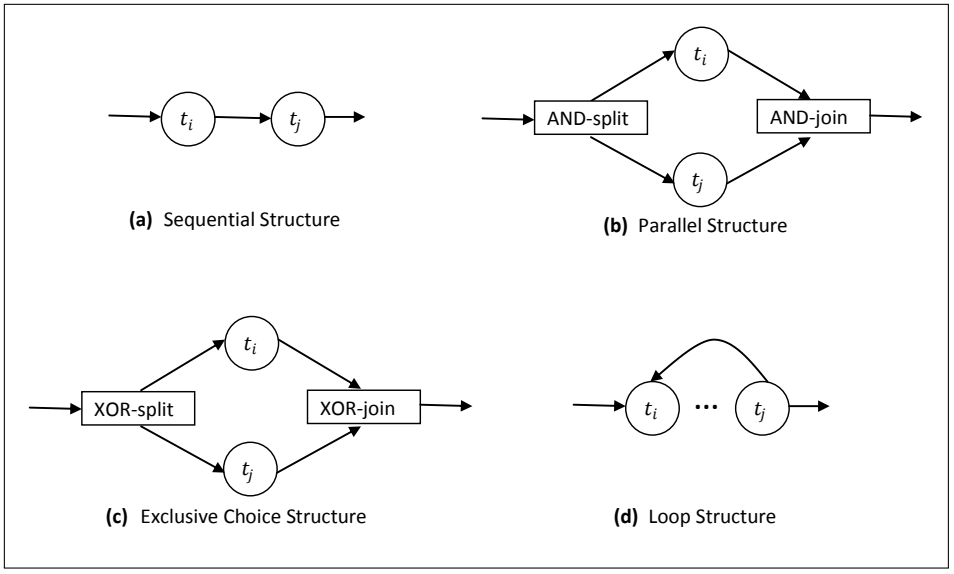

FIGURE 2.2: Common Workflow Structures

languages [4], Petri-net-based languages [129, 130, 7], and script-based languages [8, 43]. A popular example of the latter is the Business Process Execution Language[1] (BPEL), which is a standard language for managing web service composition. In order to accommodate complex workflows, many languages support nested modeling [137, 60, 120], where a top-level workflow can be recursively decomposed into sub-workflows, with atomic activities being at the lowest level.

2.4 QoS-based Service Selection

Services advertised by different providers can overlap in their functional capabilities, but offer varying quality of service (QoS) levels. It is thus important to take such QoS features into consideration when selecting services for the tasks of an abstract workflow, in order to accommodate particular user needs, or to achieve competitive advantages in current demanding markets.

[1] `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`

A QoS-aware service selection problem normally involves producing a concrete workflow that optimises some quality dimensions, while satisfying a number of quality constraints. When more than one quality dimension needs to be optimised, these are usually aggregated into an overall quality score (or utility), e.g. by applying a weighted sum on the values of these dimensions, so that the problem is treated as constrained, single-objective optimisation. Generally, current QoS-aware selection approaches can be categoriesed into local optimal selection, and global optimal selection, which are discussed below.

2.4.1 Local Optimal Selection

According to a local optimisation strategy, the best service for each abstract task in a workflow is chosen independently of other tasks. That is, given a set of functionally-equivalent services for a task, the service with the highest overall utility score in this set is selected for instantiating the task [34, 80, 136]. Although this strategy achieves high performance in terms of computation time (its complexity is linear in the number of services per task), it can only handle quality constraints at a local level (i.e. constraints on single tasks), without the ability to verify workflow-level (end-to-end or global) quality constraints, such as total price and total execution time. To illustrate, consider the simple sequential workflow shown in Figure 2.3, where two suitable services are available for each task, but with varying costs and durations. Now, suppose that the goal is to minimise the overall execution time, while keeping the overall cost below 40. Applying a local optimisation strategy entails selecting the fastest service for each task. As a result, service combination s_2s_4 is regarded as the optimal assignment in this case, which violates the imposed price constraint.

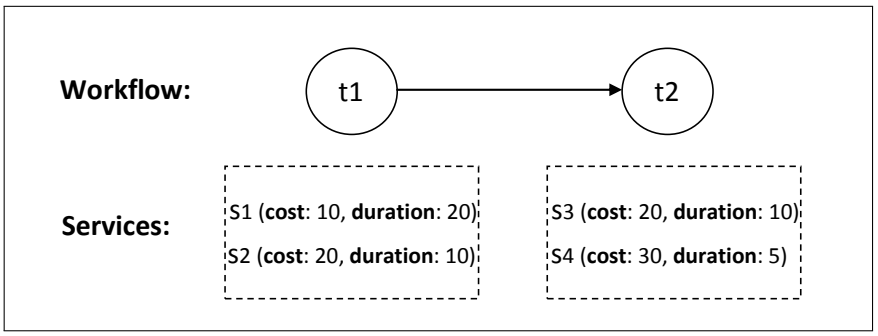

FIGURE 2.3: A simple sequential workflow with candidate services

TABLE 2.1: Possible service combinations for the workflow of Figure 2.3

Service Combination	**Total Cost**	**Total Duration**
$s_1 s_3$	30	30
$s_1 s_4$	40	25
$s_2 s_3$	40	20
$s_2 s_4$	50	15

2.4.2 Global Optimal Selection

To address end-to-end quality constraints, a global optimisation strategy reasons about service assignments from a global perspective [2]. Specifically, workflow-level QoS values are obtained for each possible service combination using suitable aggregation metrics on the component services [33]. The aggregated overall utility value is also computed correspondingly. Based on this, the optimal selection is that maximising the aggregated utility score while satisfying the end-to-end quality constraints, among all service combinations. For example, Table 2.1 shows the aggregated cost and duration values for the possible service combinations of Figure 2.3, according to which the optimal assignment for the previous example problem, *minimise total duration and ensure total cost* ≤ 40, is combination $s_2 s_3$ (rather than combination $s_2 s_4$ produced by the local optimisation strategy).

Clearly, solving the global optimal selection problem requires considering a potentially

very large number of service combinations. Since an exhaustive search of all such combinations is thus impractical, several approaches attempting to achieve better scalability have been proposed to solve this problem.

A popular way to tackle the global optimal service selection problem is by adopting Integer Programming (IP) techniques [145, 146, 13, 14]. Alternative models for the same problem, as a multi-dimension multi-choice knapsack problem (MMKP) and as a multi-constrained optimal path problem (also known as QoS routing problem), are introduced by Yu et al. [143]. These approaches, however, suffer from poor scalability when the search space of the problem increases.

To reduce the complexity of computing exact solutions, several heuristic algorithms have been employed, which find acceptable solutions in a reasonable amount of time. Yu et al. [143] propose two heuristic algorithms to solve the global optimal selection problem, one for the combinatorial MMKP model and another for the QoS routing model. Another QoS-routing based heuristic, the H_MCOP heuristic, is presented by Li et al. [76]. Alternatively, others take a genetic algorithm approach [31, 38]. Local-optimisation-based heuristics have also been proposed [6, 101], which attempt to decompose the global QoS constraints into representative local (task-level) constraints, in order to obtain an easier version of the problem. Finally, some efforts control the number of services considered for tasks, and hence reduce complexity, by following an iterative selection approach [88, 127]. The idea here is to incrementally increase the number of services at each iteration, either by a fixed number per task [127], or by choosing an alternative assignment for one task that better approximates the imposed quality criteria [88], until an acceptable solution is identified.

Despite the improvement in computation time achieved by heuristic selection methods, these methods either still do not scale well in large-scale problems, or significantly affect

the ability to find an optimal solution. Hence, finding a globally optimal service selection in an efficient way remains an open challenge. Furthermore, all current selection efforts focus on the scalability issue, ignoring the dynamism inherent in service-based systems (i.e. the service landscape is assumed to be static during the selection process), which can result in producing invalid selections. Therefore, in this thesis, we add to the existing selection attempts by contributing towards improving the efficiency of service selection. We also overcome the static-world limitation in these attempts via accounting for various types of service changes while performing selection, including service additions, service deletions, and changes in the quality values of services.

2.5 Handling Execution-time Dynamism and Uncertainty

Service-based systems rely on external heterogeneous services, distributed across the network, and owned by autonomous providers. Hence, such systems are subject to high degrees of change and volatility [74]. Run-time exceptions can occur due to various reasons including network failures, overloaded services, or unreliable providers. These can lead to unavailable components on invocation, or faulty and violating executions. Moreover, services undergo different changes during their life cycles [126, 97], driven by factors such as competition. As a result, existing service versions may be withdrawn from the market while new versions, possibly with updated functionality or different quality features, may be added, without notification. Studying the possible changes in the behaviour of web services during their life cycle, Treiber et al. [126] provide an example that a change of requirements (triggered, for instance, by the service provider) or a degradation in the service's run-time performance (for instance, due to server or network load), may lead to corresponding changes in the service's implementation (to update its functionality or to optimise its performance, respectively), thus possibly affecting the service's interface and QoS values. Given this dynamic and error-prone

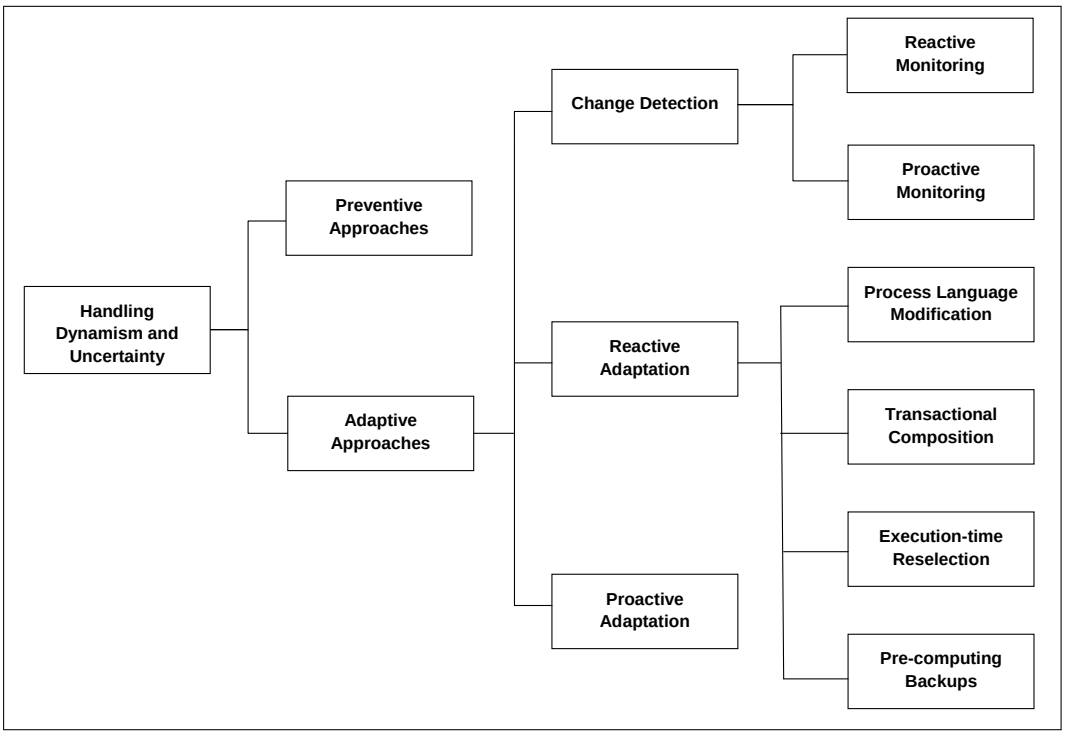

FIGURE 2.4: Categorisation of existing approaches to handling dynamism and uncertainty of service environments

nature of service-based systems, managing execution-time changes and exceptions is a vital requirement.

State-of-the-art approaches to handling dynamism and uncertainty in service-oriented systems can be categorised into *preventive* approaches, which are aimed at fault-avoidance, and *adaptive* approaches, which are aimed at fault-tolerance (see Figure 2.4 for a detailed categorisation). These approaches are discussed next.

2.5.1 Preventive Approaches

As stated earlier, the dynamism and uncertainty inherent in service-based systems result in composite applications being subjected to various types of exceptions during their execution (e.g. unfulfilled quality promises, or component unavailability). The

causes for such exceptions range from unintentional sudden hardware crashes to intentional malicious acts by self-interested service providers. When encountered, these can lead to highly undesirable situations (e.g. money loss), and demand costly corrective actions. In response, some efforts in the literature focus on introducing preventive measures to avoid such situations as much as possible, and minimise the need for execution-time adaptation.

One possible way of reducing task failures during execution is through the redundant execution of services [9, 65, 114]. That is, instead of assigning an abstract task to a single service, multiple services are invoked in parallel for the same task, in order to improve the overall success rate and duration. This approach, however, incurs increased cost due to the need for several service invocations per task, which is not always appropriate, especially in situations where the user is willing to accept the risk of unreliable execution, in favor of minimising price. Moreover, such redundancy-based approaches do not eliminate the possibility of failure, but simply reduce it.

Alternatively, some efforts focus on providing an accurate estimation of service quality values, as an attempt to minimise quality deviations at run time. For example, Ivanovic et al. [63] present a data-driven approach for producing more accurate quality predictions. The idea is to define the upper and lower bounds of a service's computation cost as functions of the input data in order to increase the estimation precision, with these functions being derived by applying existing tools to the logical translation of the service's BPEL process. Trust and reputation mechanisms have also been considered for the purpose of accurate predictions [84, 85, 132, 77, 100]. In particular, prior to an interaction with a service, an assessment of its reputation or the trustworthiness of its QoS advertisements (which are claimed by the provider) is undertaken, in order to avoid selecting unreliable services that may not honour their promises. Such an assessment is usually based on the available information regarding the past performance of

the service which, for example, could be obtained as feedback from its users, or shared by the software agents that previously interacted with the service. However, these prediction attempts cannot ensure guaranteed compliance of a service at execution time. Additionally, exceptions could still occur for various other reasons, either intentional (e.g., because the service provider changes the quality values offered by its service) or unintentional (e.g., due to network, hardware, or software problems).

To sum up, although the above approaches contribute towards reducing quality deviations and failures during execution, complete avoidance of execution-time exceptions is not possible. Hence, the ability to adapt to changes remains a critical requirement for systems operating in dynamic service environments.

2.5.2 Adaptive Approaches

Achieving fault-tolerant behaviour in a system usually requires two capabilities [17], exception detection and corresponding system recovery. Depending on the type of the exception and the time of its detection, recovery actions could either be *reactive*, to handle an encountered error, or *proactive*, to prevent future failure or to further improve performance. Current approaches to change detection, reactive adaptation, and proactive adaptation, in the context of service-based systems, are introduced next.

2.5.2.1 Change Detection

The first step to achieving adaptive and fault-tolerant behaviour is the *detection* of unusual situations (changes in the environment) that may need special treatment. Typically, this detection is achieved through run-time *monitoring* [47, 96], which observes the execution of services comprising an application, and if the actual behaviour of a service does not comply with that expected, an adaptation need is identified. With

such *reactive* monitoring, violations and faulty actions are detected only after their execution [94], causing undesired effects and possibly unrecoverable situations. It is therefore preferable that service changes are detected as early as possible, to reduce or even prevent any damage and facilitate better adaptation opportunities. Run-time failure prediction has consequently gained increasing interest, allowing identification of future problems before their actual occurrence, through *proactive* monitoring of the current state of the system [107].

For example, a number of approaches are concerned with modelling volatility in service response time, very often caused by the network. In this regard, Dai et al. [44] and Yang et al. [139] predict changes in data transmission time (and consequently service response time) for the non-executed component services of a selected solution, through a Semi-Markov Process, in centralised and decentralised composition systems, respectively. Similarly, Aschoff et al. [15] model the response time of a service as a random variable, changing as a result of various factors related to the network and system resources (e.g. request queuing time). The exponentially weighted moving average is utilised for estimating the expected value of this variable at a particular time step, based on historical data.

Other approaches adopt online testing techniques in order to proactively discover service unavailability and failures (functionality-based or QoS-based) at run time, before these cause unwanted effects. In the PROSA (PRO-active Self-Adaptation) framework [58], the behaviour of services is tested using generated test cases. Such testing is performed, e.g. periodically, in parallel with the execution process, and if the behaviour observed in the test differs from that expected, an adaptation request is identified. Tosi et al. [123] specify nine fault categories related to various service problems (e.g. functional, non-functional, technical), serving as guidance for designers to identify suitable test cases to be triggered on the services discovered at run time. Metzger et al. [90]

and Sammodi et al. [108] integrate monitoring-date-based failure prediction with online testing in order to improve prediction accuracy, thus avoiding unnecessary adaptation requests. Specifically, in the case where the relevant data samples observed so far regarding a service are not sufficient for prediction purposes, test cases are generated and instantiated to increase the number of these samples. Normally, testing-based approaches assume that services either have no side effects, or expose a special mode for testing purposes.

Two run-time service discovery modes, namely the pull and push modes, are proposed by Zisman et al. [149]. The traditional pull mode involves querying the service registry reactively for alternatives in response to service misbehaviour during execution. The push mode, on the other hand, is performed proactively, in parallel with the execution process, to allow the early detection of both problematic (e.g. changes in the candidate service characteristics) and non-problematic (e.g. the availability of a new candidate service) changes. This is achieved through subscribing to requests for such changes, with the registry notifying the corresponding service listener on their occurrence.

In this thesis, we assume the ability to detect changes in the environment, and focus on the subsequent steps concerning the proper handling of such changes. That is, our work is concerned with the effective analysis of these changes, the identification of the corresponding adaptation actions that need be taken to maintain composition's optimality and compliance with user constraints, and the execution of these actions efficiently and as transparently as possible (from the end user perspective). The above detection approaches can therefore be considered complementary to our work.

2.5.2.2 Reactive Adaptation

At the basic level, a fault tolerant system should be able to continue its intended execution, or at least terminate in a consistent state, in spite of the occurrence of failure

or violation. This is usually achieved through appropriate corrective actions that are triggered in reaction to erroneous behaviour, and is referred to as reactive adaptation. Most current adaptive service composition approaches fall under this category. These approaches are discussed next.

2.5.2.2.1 Process Language Modification A number of approaches to fault tolerance propose incorporating fault handling mechanisms into the workflow modeling language itself. That is, rather than fully depending on the run time environment, the workflow designer (or user) is provided with a means to control the adaptation actions in response to exceptions during execution, thus achieving more reliable recovery. Examples of such approaches are presented below.

Similarly to programming languages, Hagen et al. [57] add exception handling constructs to the workflow process specification of OPERA's process support system, the semantics of which is as follows. When a task fails, it signals an exception to its parent, with the possible actions by the latter's exception handler including: resume the failed task after resolving the error, e.g. by asking the user for valid data; abort and replace the failed task with the application of appropriate compensation; or propagate the error up in the process hierarchy if it is unsolvable at the current level.

Colombo et al. [40] introduce a composition language, and a corresponding execution platform, supporting the explicit specification of recovery strategies in response to failures and violations at execution time. Specifically, the traditional BPEL workflow is extended with rules that allow the designer to guide re-configuration actions at run time. This rule-base language is interpreted and executed by the SCENE (Service Composition ExecutioN Environment) platform.

Alternatively, Hoheisel [60] integrates user-defined fault handling strategies into a workflow modeling language based on Petri nets. Another Petri-net-based extension is

proposed by Tolosana-Calasanz et al. [121], where exception handling capabilities are incorporated into the workflow language of DVega's scientific workflow system, which utilises the reference nets formalism. According to this extension, the user can equip each workflow node with two possible actions for addressing a particular exception at the node during execution: propagating the exception to a higher level in the workflow hierarchy where it can be handled successfully, or replacing the workflow node with a suitable alternative. The latter is achieved with respect to three selection strategies: exception-context based, price-based, and execution-time based.

Although effective for dealing with specific types of execution-time exceptions, e.g. invalid input or output parameters/assertions, which could be better treated by utilising designer (or user) expertise, such process-integrated adaptation may not be suitable for some other types. These include service changes requiring early proactive actions (e.g. unavailability or changes in the quality values of a service to be executed in the future), or those emerging new opportunities not directly concerning the process instance being executed, but affecting optimality. Accommodating such cases within the process specification is either not possible, or results in an explosion of the exception handler complexities. Moreover, the designer may simply not be aware of such changes in advance. Therefore, in this thesis, we achieve the required adaptive behaviour at the middleware level, allowing more flexible and comprehensive management of service changes (additions, deletions, or changes in quality values) in the dynamic and uncertain settings we consider. Such management also involves accounting for the complexity of the multi-constrained optimal re-selecion of services, not addressed by the above approaches, where replacements are either pre-determined at design time, or selected at run time using simple, single criterion strategies.

2.5.2.2.2 Transactional Composition A number of approaches consider incorporating transactional support into the service composition process in order to increase

composition reliability and fault tolerance at execution time [91, 27, 28, 79, 76, 55]. These efforts aim to minimise the risk for consumers by ensuring that the execution terminates in a consistent state even when failures occur, achieved through compensation policies allowing the effects of executed services to be undone. To support this, a number of web service standards have also been proposed [42] (e.g., WS-Coordination[2] and WS-AtomicTransaction[3]).

The main idea in such approaches is satisfying a particular transactional behaviour by the produced composite service (viewed as a transaction unit). This behaviour is either integrated into the workflow as transactional patterns [27, 28, 79], or imposed by the user as transactional requirements [91, 55]. Generally, the transactional behaviour of a composite service is determined by the transactional properties of its comprising components, which are commonly classified into three kinds [27, 28, 76, 55]: pivot, compensatable, and retriable. A service is *pivot* if, once completed, its effects are permanent, but it has no effects in the case of failure. A service is *compensatable* if its effects can be eliminated, possibly through another service. Finally, a service is *retriable* if it ensures successful completion after several finite invocation attempts. Based on this, achieving atomic behaviour, for example, by a composition, requires ensuring that all the component services preceding a non-retriable service (i.e. an error-prone service) are compensatable (not pivot).

Despite providing consistent and more reliable process execution, transactional approaches offer rather extreme and costly exception-handling capabilities, which may not be necessary in many situations. For instance, instead of backtracking all previous steps in the currently executed process instance as a result of a service failure, it may be cheaper and more desirable simply to replace the failed service with an available

[2] `http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-errata-os/wstx-wscoor-1.1-spec-errata-os.html`

[3] `http://specs.xmlsoap.org/ws/2004/10/wsat/wsat1104.pdf`

alternative, and continue execution. Moreover, these approaches are constrained to cooperative environments, and do not take preventive actions concerning early (pre-invocation) service unavailability nor account for emerging better opportunities. Nevertheless, they can be considered as an interesting complement to our execution-time change handling approach, which will mainly focus on forward recovery (i.e. all services are considered pivot, without allowing compensation possibilities).

2.5.2.2.3 Execution-time Re-selection A popular way of recovering from unexpected situations at execution time is by triggering re-planning actions in response. Specifically, services are re-assigned for the remaining, non-executed tasks in a way that guarantees still meeting the end goals, in spite of the erroneous behaviour already executed. The newly-selected service plan is then adopted to continue the execution process.

Some such efforts apply, during the re-planning stage, the same selection algorithm used to produce the initial solution, but incorporating the current execution status. For example, Zeng et al. [146] recalculate assignments for the non-executed part of a workflow each time a change occurs during execution by adopting Integer Programming. A re-planning triggering algorithm is introduced by Canfora et al. [30], to recalculate quality values of a composite service according to the new information at execution time (e.g., actual service qualities, or actual number of loop iterations), and if the new qualities differ considerably from previously estimated ones, execution is stopped and a genetic-algorithm-based re-planning is triggered for remaining workflow tasks. A similar execution-time re-planning approach, but based on Integer Programming, is presented by Ardagna et al [14], where re-selection of services for the non-executed tasks is performed in response to one of the following: a considerable deviation in the actual quality values from those expected; a failure in a component service invocation; changes in the user requirements; or availability of information regarding the number

of loop iterations or the conditional branch to be executed. In addition, re-selection is also triggered periodically, with the re-selection period being set according to the dynamism of the environment.

Others introduce heuristic methods for the re-selection process to reduce its computational complexity, especially that such re-selection takes place while the application is running. For example, Berbner et al. [25] use the H1_RELAX_IP heuristic, backtracking on the results of a relaxed integer program, to re-plan the remaining part of the workflow in a timely manner, when the actual quality values of an executed service differ from those expected or the component service to be invoked is unavailable. Likewise, to recover from faulty services at execution-time, Lin et al. [78] propose a region-based heuristic re-selection algorithm, which iteratively expands the sub-process to be reconfigured until a satisfactory replacement is found.

In summary, the reactive re-selection approach to adaptation allows a composite application to continue its intended execution, even in the presence of violations. However, this approach has several limitations. First, it causes an interruption to the composite service execution until the costly re-selection is performed, which could be highly undesirable, especially in the case of time-sensitive applications. Moreover, the late reaction to changes (i.e. after faulty or quality violating services have been executed), might result in an inability to find a suitable recovery from that point, or a re-selected solution of lower quality compared to one that could be obtained when reacting to changes earlier. Finally, such an approach is mainly corrective, and does not account for emerging better opportunities, which improve solution quality when considered.

2.5.2.2.4 Backup Approaches In order to eliminate the high re-selection overhead in the face of component failures at execution time, some approaches consider

supporting the composite application with backup services to ensure its continuous execution, utilising functional equivalence among services [19, 18].

Yu et al. [141, 142] propose to compute, during selection, a back-up composite service for each component in the selected solution (i.e. the best combination of services for the remaining part, without considering the service currently selected for the next task). In this way, the back-up solution of the current component service can be used to continue execution without any extra delay in the case where the next component service to be invoked according to the original solution is unavailable. This approach is designed to handle only a single service failure at execution time (multiple failures cannot be accommodated by this approach).

Similarly, Chafle et al. [35] suggest preselecting a set of alternative solutions for the composition problem, so that whenever changes in the environment affect the currently executing solution, it is replaced with the best alternative satisfying the quality constraints.

Wagner et al. [133] identify multiple backup services for each task of the workflow during the initial selection process. For this purpose, functional clustering is first applied on services to determine substitutable alternatives. Based on this, when an originally-selected service fails, it can be replaced by a service from its sub-cluster (the one with the shortest distance to the original service). An interesting feature of this approach is that the backup services are taken into consideration during the QoS-based optimal selection process, allowing the estimation of the quality characteristics of compositions in the case of failures beforehand, and correspondingly making appropriate selection decisions according to user risk preferences.

These approaches achieve quick and efficient adaptation in response to component failures at execution time, but do not account for continuous changes in the service landscape (addition, deletion, or changes in the quality values, of services). In other

words, since the backups are precomputed at selection time, these may no longer remain optimal, satisfactory, or even available during execution, especially in the case of long-running workflows. As a result, the execution could be faced with either a low-quality alternative, or a costly replanning process to achieve a successful (or better) recovery.

2.5.2.3 Proactive Adaptation

As opposed to reactive adaptation where the system adapts itself after a violation has occurred, proactive adaptation attempts to perform reconfiguration actions ahead of time, before reaching problematic execution points. Such early adaptation facilitates better recovery options, and avoids undesirable situations such as the invocation of faulty services and overall performance degradation due to execution disruptions to find replacements. Proactive adaptation could also be instantiated for optimisation purposes, to exploit newly available opportunities, thus improving system characteristics and increasing user satisfaction. Current approaches to proactive adaptation, however, are still very limited.

In light of proactive change detection, a number of efforts advocate the need for corresponding proactive adaptation strategies [149, 58, 123, 90, 108, 89]. Such strategies, for example, could be assigned to the output of proactive test cases performed on services [58, 123], or triggered when the predicted failure probability of a component exceeds a particular threshold [90]. However, these efforts provide no concrete details concerning these strategies and their actual execution.

Based on response-time prediction, Dai et al. [44] and Yang et al. [139] propose the instantiation of a quality-based re-selection process to produce a backup solution, whenever a deviation at a non-executed component is anticipated, and this component's pre-computed replacement is not available. The execution could then quickly switch to this backup when the failure occurs. Despite the early reaction to changes in both

approaches (which reduces interruption time), a number of limitations can be identified. First, the re-selection only concerns the affected task and its successors. However, better (and sometimes the only possible) solutions could be obtained by also reselecting services for the non-executed predecessors of the affected task. Moreover, this re-selection is only triggered for repair purposes, thus neglecting optimisation possibilities. Finally, only changes in response time are handled, without addressing other quality attributes.

Aschoff et al. [15, 16] propose a simple adaptation algorithm to be triggered when a problem at a component is predicted, as follows. First, the algorithm tries to find a suitable replacement for this component from its candidate set. If no alternative can be found without violating the quality constraints, the re-selection region is incrementally increased, traversing the candidate sets of the other non-executed components in order, until a satisfactory composition is identified. However, neither the quality optimisation problem, nor the issue of execution interruption that could be caused by the re-selection process, is considered.

Rather than initially optimising a combination of services for a workflow's tasks, Stein et al. [115] optimise a combination of meta-level strategies (based on a local search heuristic), specifying suitable, local-level selection and re-selection policies for each task. Such meta-planning allows the delay of the actual selection of services until the task is due, thus implicitly accommodating all the changes in the service landscape prior to task invocation. However, the strategy allocation, which is continuously updated on the availability of new information at execution time, mainly depends on average performance statistics about each task, and thus cannot cope with end-to-end quality constraints. This is because these constraints require reasoning about the actual QoS values of available service combinations, and thus the average-performance-based reasoning is not sufficient to ensure their satisfaction.

Overall, the majority of existing proactive adaptation attempts mainly focus on the change detection part, giving little consideration to the actual adaptation process in response to such detection. Specifically, the adaptation actions are either omitted or provided as simple replacement polices, without addressing the quality-based global optimal selection problem. When considered, these actions are mostly instantiated for corrective purposes, to prevent an anticipated problem, ignoring changes such as additions of new services for the non-executed tasks that do not directly affect the currently selected composite service, but might result in a much better solution in terms of optimality. Furthermore, no proper modeling and management of the adaptation process, to avoid its interference with the application's execution, is provided. These issues are addressed in this thesis.

2.6 Handling Inter-Service Dependencies

Despite the autonomous nature of the services participating in service-based systems, these services still experience various types of dependencies [131]. Hence, accounting for such dependencies when aggregating services into higher-level applications is essential to ensure valid and good-quality compositions.

The most obvious type of inter-service dependency is the *flow* correlation, which constrains the execution order of services in a composition. For instance, such a correlation could be imposed by the business logic, or be due to the dependence of one service on data or resources produced by another. In the context of workflow-based service composition, flow correlations can easily be captured by the workflow structure (see Section 2.3.2).

Another type of correlation that needs to be accounted for among services is the *compatibility* correlation, which declares whether two services can work together in a composition, without errors. A common reason for a composability problem is a semantic mismatch between service interfaces [54, 75]. This is usually discovered by calculating the semantic similarity degree between the input and output parameters of services, to assess how well these services fit together. Another reason could be provider business policies, which restrict the usage of some services to limited combination scenarios [76, 138]. In this thesis, we do not consider such incompatibility issues among services.

Quality correlation represents a third type of dependency that could exist between services, where the quality values offered by a service depend on the services co-selected with this service in a composition [54, 118, 138]. This dependency may occur, for instance, due to business alliances, technological compatibilities, etc. Inter-service QoS correlations can considerably influence the quality features of service compositions, and are thus directly relevant to the global optimal service selection problem of interest in this thesis. In particular, accounting for such correlations during the selection process allows production of more accurate quality estimations for service compositions, and consequently making correct selection decisions. Yet, despite their importance, these correlations are generally ignored in the current service selection literature, with only a few recent attempts being introduced to capture them, as summarised below.

Zhang et al. [148] propose mining inter-service correlations from historical information about service usage. The idea is to discover *implicit* correlation patterns that achieved good performance in the past, and to utilise these to improve the quality of selected composite solutions. However, this approach does not support any *explicit* modelling or reasoning capabilities concerning quality correlations.

Guo et al. [54] and Wu et al. [138] reflect inter-service correlations explicitly in the quality model of services. The representation in both approaches, however, is very simple and limited to binary correlation cases, thus failing to represent more complex, real-life scenarios.

Finally, a richer correlation-aware QoS model for resource services is introduced by Tao et al. [118], and a resource composition method based on particle swarm optimisation is suggested. However, no special consideration is given to the efficiency issues of selection in this approach.

To summarise, awareness of quality correlations poses additional challenges to the service selection problem, and these have yet to be addressed. In particular, the traditional, dependency-free, quality model of services needs to be augmented with additional expressive power to accommodate various, possibly complex, quality dependencies among services. Achieving selection scalability in the presence of such correlations is another issue calling for suitable solutions, especially because the traditional efficiency-boosting mechanisms developed in the context of correlation-ignorant search spaces may no longer be applicable for more complex, correlation-aware spaces.

2.7 Conclusion

Enabling adaptive and QoS-aware dynamic service composition has received a great deal of attention in recent years, but is still far from being satisfactory. In particular, the following limitations can be identified with respect to the research challenges set in Chapter 1.

Granular Heterogeneity. The skeleton of a composite application is usually specified by means of an abstract workflow, identifying the flow of atomic tasks to be accomplished.

At run time, the services sought for implementing the application are only those satisfying the functional constraints of these tasks, possibly neglecting other relevant services available at different granularity levels (e.g. special packages or service bundles [3]). Despite some existing suggestions for a hierarchical workflow representation, this is mainly for the purpose of simplifying the modelling process of complex workflows, and the atomic activities remain the only ones instantiated to actual services. For example, Hierarchical Task Network (HTN) planning [111] is a popular planning methodology utilising the concept of task decomposition to achieve a particular planning goal (by recursively decomposing tasks into finer-grained ones according to pre-defined methods), but only primitive tasks are assigned to concrete planning operators (services).

Time Constraints. Several exact and heuristic methods have been proposed to solve the global optimal service selection problem (see Table 2.2), both for the initial (pre-execution) planning and the later (execution-time) re-planning stages. These methods, however, either jeopardise the quality of the solution produced, or suffer from efficiency issues in a large-scale SOA. Poor efficiency is highly undesirable, especially at the re-planning stage that takes place in current approaches *after* the occurrence of an exception, and hence can cause long interruptions to the execution process.

Dynamism and Uncertainty. Since total avoidance of faults and violations in the open and distributed service-based settings is almost impossible, tolerance against such exceptions at execution time is essential. Many proposals have been provided in this regard (see Table 2.3). These, however, can mostly be classified as reactive, performing corrective actions only after an exception has already occurred, thus lacking any ability to avoid erroneous behaviour or improve performance, when possible. In addition, an interruption to the execution process is incurred until the corrective actions (usually through costly re-planning) are completed. Attempts to reduce such interruption include applying fast heuristics or pre-computing backup plans beforehand. While the

TABLE 2.2: Summary of QoS-based Global Selection Approaches

	Solution Optimality	**Scalability with Increasing Size**	**Selection-time Reactivity**	**Inter-service QoS Correlation Awareness**
Exact Solutions:				
Zeng et al. [145, 146]	✓	✗	✗	✗
Ardagna et al. [13, 14]	✓	✗	✗	✗
Yu et al. [143]	✓	✗	✗	✗
Heuristic Solutions:				
Yu et al. [143]	✗	to some extent	✗	✗
Canfora et al. [31]	✗	to some extent	✗	✗
Alrifai et al. [6]	✗	✓	✗	✗
Qi et al. [101]	✗	✓	✗	✗
Menasce et al. [88]	✗	✓	✗	✗
Tao et al. [118]	✗	to some extent	✗	✓
Guo et al. [54]				only simple binary cases

former affects the solution quality, the constant changes of the service landscape may invalidate the latter, making the backups no longer applicable or poor-quality choices. Despite some recent efforts on proactive adaptation, these mostly focus on the exception prediction step, ignoring the actual adaptation process. Moreover, while it is important to take early proactive actions starting from the selection stage, selection-time changes are neglected by current research (see Table 2.2), risking initial invalid solutions in the first place.

Service Correlations. Existing attempts to QoS correlation awareness are largely limited, with most service selection approaches assuming quality independence among services (see Table 2.2). This is likely to result in inaccurate QoS estimations for compositions, and consequently poor-quality selected solutions.

In this thesis, we seek to overcome the above limitations through the chapters presented next. In particular, to accommodate the granular heterogeneity of services, we guide the selection process by a rich collection of alternative planning options, incorporating functional abstractions at various levels of granularity. Moreover, to boost selection efficiency, we develop pruning techniques that clear from the search space unpromising

TABLE 2.3: Summary of Execution-time Adaptive Composition Approaches

	Reactive Recovery	**Seizing Emerging Opportunities**	**Preventive Recovery**	**Solution Optimality**	**Avoiding Execution Disruption**
Reactive Attempts: Reselection					
Ardagna et al [14]	✓	✗	✗	✓	✗
Canfora et al. [30]	✓	✗	✗	✗	✗
Berbner et al. [25]	✓	✗	✗	✗	✗ (but reduced)
Lin et al. [78]	✓	✗	✗	✗	✗ (but reduced)
Reactive Attempts: Precomputed backups					
Yu et al. [141, 142]	one failure	✗	✗	✗	✓
Wagner et al. [133]	✓	✗	✗	✗	✓
Proactive Attempts:					
Dai et al. [44]	✓	✗	✓	✗	partially
Yang et al. [139]	✓	✗	✓	✗	partially
Aschoff et al. [15, 16]	✓	✗	✓	✗	partially

choices without any impact on optimality. Furthermore, we achieve adaptive system behaviour at two stages, the composition selection stage and the composition execution stage, through novel reactive selection and execution algorithms, respectively. While the former allows an initially valid, optimal, and satisfactory solution, the latter ensures that these characteristics are retained while execution is progressing, in spite of high environment volatility. A key feature of this adaptive behaviour is the *early* reaction to changes via *efficient* corrective and optimising actions, preformed without interfering with the execution process. Finally, we accommodate various, possibly complex, cases of quality dependencies among services, and provide corresponding techniques to prune the search space and ensure a scalable selection in their presence.

Chapter 3

Multi-Granularity Service Selection Model

3.1 Introduction

Composite applications are usually specified as abstract workflows, defining the outline of atomic tasks to be accomplished. At run time, these tasks are allocated to actual services based on service availability and the quality of service (QoS) criteria of the user. Such specifications, however, limit the candidate services for composition to those that satisfy the functional granularity requirements of the workflow's atomic tasks, neglecting relevant services with different granularity. For instance, two atomic tasks might be better achieved with a coarse-grained service that combines their functionalities than with two individual finer-grained ones (e.g., due to the better quality of the service or the reduced communication overhead). To illustrate, consider a scenario in which the user wants to go on holiday, while minimising the overall price regardless of other qualities. To achieve this, she contacts a travel software agent with the request *plan*

holiday, for which the agent generates a plan with the following sub-tasks: *book flights*, *book hotel*, *book sightseeing*, and *apply for visa*. The agent searches for the cheapest services to execute each subtask of the generated plan, which are (for instance) services s_a, s_b, s_c, and s_d, respectively. Now, suppose a service s_e combines the two functionalities of booking a hotel and booking sightseeing with a special offer on price. Here, the generated composite service (s_a, s_b, s_c, s_d) might not be the cheapest, and replacing s_b and s_c with s_e might result in a composite service with better quality (cheaper price).

In response, this chapter proposes a richer representation of the compositional knowledge that allows the expression of not only alternative composition plans, but also the different hierarchical levels of the tasks involved. Based on this hierarchical representation, a formal model of the service selection problem, of interest in this thesis, is provided. The model adopts the following notation (which is used henceforth): upper-case identifiers to denote sets; lower-case identifiers to denote functions and variables; identifiers with an upper-case first letter to denote relations; and a pair (V, E) to represent a graph, where V is the set of nodes (vertices), and E is the set of edges.

The rest of the chapter is organised as follows. The hierarchical planning knowledge representation is introduced in Section 3.2. Section 3.3 models the non-functional properties of services, followed by the service model in Section 3.4. The planning knowledge tasks are mapped to concrete services through the service discovery model in Section 3.5, while the non-functional properties of the resulting composite services are modelled in Section 3.6. Sections 3.7 and 3.8 present the request model and the service selection problem, respectively. Finally, Section 3.9 concludes the chapter.

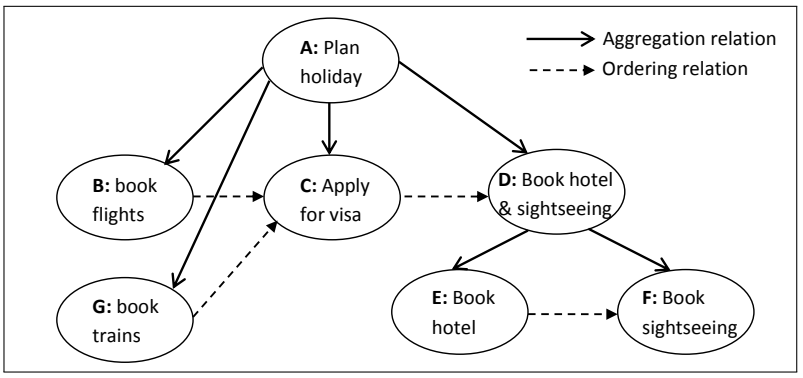

FIGURE 3.1: Planning knowledge hierarchy for *plan holiday* task

3.2 Planning Knowledge Model

The planning knowledge for a particular objective can be represented as a hierarchy of tasks, with the root being the goal task to be achieved. Each task is annotated with semantic descriptions specifying the functional requirements of the suitable services for this task, e.g., in terms of OWL-S inputs, outputs, preconditions, and effects. The sub-tasks (of each composite task) are partially ordered, and thus can be modelled as a directed acyclic graph (*dag*). In addition, there may be more than one way to decompose a task, resulting in alternative decomposition graphs. Note the we assume the *functional validity* of all decomposition graphs, which could, for instance, be pre-determined by a human expert. An example *planning knowledge hierarchy* for the goal task *plan holiday* is shown in Figure 3.1.

Formally, the planning knowledge hierarchy can be represented as a quintuple:

$$(T, Taggr, root_t, func_t, graph_t)$$

where the components are as follows.

- T is a finite set of the tasks involved;

- $Taggr$ is an aggregation relation between tasks, hierarchically decomposing coarse-grained tasks into finer-grained ones:

$$Taggr \subset T \times T$$

where $\langle t_i, t_j \rangle \in Taggr$ indicates that t_i is the aggregating (parent) task while t_j is the aggregated (child) task.

- $root_t$ is the root of the hierarchy, representing the goal task to be achieved, such that:

$$\forall t \in T, \quad \langle t, root_t \rangle \notin Taggr$$

- $func_t$ is a functionality description function, which assigns to each task $t \in T$ a semantic specification of its functional requirements:

$$func_t : T \to FD$$

where FD is the set of all functionality descriptions.

- $graph_t$ is a functionality decomposition function, which maps each non-leaf task $t \in T$ to a set of *dag*s, each representing an alternative decomposition of task t:

$$graph_t : T \to 2^G$$

where G is the set of all *dag*s that can be formed from the tasks in T, and $\forall t \in T,\ \forall g = (V, E) \in graph_t(t), \quad g$ satisfies the following:

  - $V \subset T$, such that $\forall v \in V, \quad \langle t, v \rangle \in Taggr$; and
  - $E \subset V \times V$ is a strict partial order on the tasks in V, with $\langle v_i, v_j \rangle \in E$ indicating that task v_i should be executed before task v_j.

Example. The planning knowledge of Figure 3.1 can be represented as a tuple $(T, Taggr, root_t, func_t, graph_t)$:

$$\begin{aligned}
T &= \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}, \mathrm{G}\} \\
Taggr &= \{\langle \mathrm{A}, \mathrm{B}\rangle, \langle \mathrm{A}, \mathrm{G}\rangle, \langle \mathrm{A}, \mathrm{C}\rangle, \langle \mathrm{A}, \mathrm{D}\rangle, \langle \mathrm{D}, \mathrm{E}\rangle, \langle \mathrm{D}, \mathrm{F}\rangle\} \\
root_t &= \mathrm{A} \\
graph_t(\mathrm{A}) &= \{(\{\mathrm{B}, \mathrm{C}, \mathrm{D}\}, \{\langle \mathrm{B}, \mathrm{C}\rangle, \langle \mathrm{C}, \mathrm{D}\rangle\}), (\{\mathrm{G}, \mathrm{C}, \mathrm{D}\}, \{\langle \mathrm{G}, \mathrm{C}\rangle, \langle \mathrm{C}, \mathrm{D}\rangle\})\} \\
graph_t(\mathrm{D}) &= \{(\{\mathrm{E}, \mathrm{F}\}, \{\langle \mathrm{E}, \mathrm{F}\rangle\})\} \\
graph_t(\mathrm{B}) &= graph_t(\mathrm{G}) = graph_t(\mathrm{C}) = graph_t(\mathrm{E}) = graph_t(\mathrm{F}) = \emptyset
\end{aligned}$$

Based on this model, alternative *abstract plans* may be available for achieving a particular task, specifying required sub-tasks and their ordering constraints. For ease of definition, we first introduce the following functions.

- Function *nodes* maps a *dag* to its corresponding nodes:

  $$nodes : G \to 2^T$$

  such that $\forall g = (V, E) \in G, \quad nodes(g) = V$

- Function *edgs* maps a *dag* to its corresponding edges:

  $$edgs : G \to 2^{T \times T}$$

  such that $\forall g = (V, E) \in G, \quad edgs(g) = E$

- Function *snode* maps a *dag* to its corresponding start nodes (those with no incoming edges):

  $$snode : G \to 2^T$$

such that $\forall g = (V, E) \in G, \quad snode(g) = \{v_i \in V \mid \forall v_j \in V, \quad \langle v_j, v_i \rangle \notin E\}$

- Function *enode* maps a *dag* to its corresponding end nodes (those with no outgoing edges):

$$enode : G \to 2^T$$

such that $\forall g = (V, E) \in G, \quad enode(g) = \{v_i \in V \mid \forall v_j \in V, \quad \langle v_i, v_j \rangle \notin E\}$

- Function *expnode* maps a *dag* to a set of graphs, each resulting from replacing a node in the original *dag* with one of its decomposition graphs (e.g. Figure 3.2(d) shows Figure 3.2(b) with node D expanded):

$$expnode : G \to 2^G$$

such that $\forall g = (V, E) \in G, \quad expnode(g) =$

$$\begin{aligned}
\{g_{exp} = (V_{exp}, E_{exp}) \in G \mid \exists v_{exp} \in V,\ \exists g_{ch} = (V_{ch}, E_{ch}) \in graph_t(v_{exp}),& \\
V_{exp} = (V / \{v_{exp}\}) \cup V_{ch} \quad \wedge& \\
E_{exp} = \bigcup_{\substack{\langle v_i, v_j \rangle \in E \\ v_i \neq v_{exp} \\ v_j \neq v_{exp}}} \{\langle v_i, v_j \rangle\} \quad \cup \quad E_{ch} \quad \cup& \\
\bigcup_{\langle v_i, v_{exp} \rangle \in E} \bigcup_{v_j \in snode(g_{ch})} \{\langle v_i, v_j \rangle\} \quad \cup& \\
\bigcup_{\langle v_{exp}, v_j \rangle \in E} \bigcup_{v_i \in enode(g_{ch})} \{\langle v_i, v_j \rangle\} \quad \}&
\end{aligned}$$

- Function *expgraph* gives all possible expansions of a graph:

$$expgraph : G \to 2^G$$

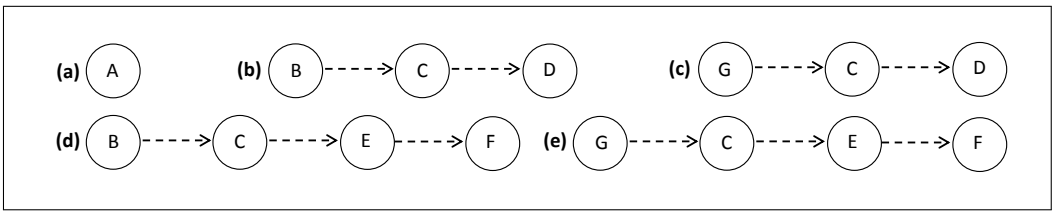

FIGURE 3.2: Alternative abstract plans for achieving task A of Figure 3.1

such that

$$\forall g \in G, \quad expgraph(g) = \{g_{exp} \in G \mid \exists g_{node} \in expnode(g), \\ g_{exp} = g_{node} \quad \vee \quad g_{exp} \in expgraph(g_{node})\}$$

Now, the alternative abstract plans for achieving a given task $t \in T$, according to the planning knowledge hierarchy, can be defined using function *plan*, which assigns to each task its corresponding abstract plans:

$$plan : T \rightarrow 2^G$$

such that $\forall t \in T, \quad plan(t) = \{g \in G \mid g = (\{t\}, \emptyset) \vee g \in expgraph((\{t\}, \emptyset))\}$. For example, task A in Figure 3.1 has five possible abstract plans, shown in Figure 3.2.

3.3 Quality Model

When multiple services offer similar functionality, they can be distinguished according to their quality of service (QoS) attributes (the non-functional properties of services). Formally, knowledge of these attributes comprises a quadruple:

$$(NAME_a, VALUE_a, dom_a, dir_a)$$

where the components are as follows.

- $NAME_a$ is the set of all quality attribute names (the non-functional properties of services). For instance, $NAME_a = \{\text{price}, \text{reliability}, ...\}$.
- $VALUE_a$ is the set of all possible quality attribute values. That is, the union of the domains of all quality attributes.
- dom_a is a domain function, which maps each quality attribute to its corresponding domain (the possible values of this attribute):

$$dom_a : NAME_a \rightarrow 2^{VALUE_a}$$

- dir_a is a direction function, which associates each quality attribute with either an *increasing* (+) direction (the quality increases as the attribute value increases) or a *decreasing* (-) direction (the quality decreases as the attribute value increases):

$$dir_a : NAME_a \rightarrow \{+, -\}$$

e.g. $dir_a(\text{reliability}) = +, \quad dir_a(\text{price}) = -$

(note that the above quality model is similar to existing models [146, 13, 143, 6]).

3.4 Service Model

The space of available services can be defined as a tuple $(S, name_s, value_s, func_s)$, where:

- S is the set of all available services;

- $name_s$ is an attribute name function, which assigns to each service the names of its quality attributes:

$$name_s : S \rightarrow 2^{NAME_a}$$

- $value_s$ is an attribute value function, which assigns to each service the values of its quality attributes:

$$value_s : S \times NAME_a \rightarrow VALUE_a \cup \{\text{undf}\}$$

such that:

$$\forall s \in S, \ (\forall a \in name_s(s), \ \ value_s(s,a) \in dom_a(a)) \wedge$$
$$(\forall a \notin name_s(s), \ \ value_s(s,a) = \text{undf})$$

Here, $value_s(s,a) = \text{undf}$ indicates that the value is not defined (not applicable).

- $func_s$ is a functionality description function, which assigns to each service a semantic specification of its functionality (e.g. in OWL-S or another semantically-enriched specification language):

$$func_s : S \rightarrow FD$$

3.5 Service Discovery Model

The candidate services for each task in the planning knowledge hierarchy can be defined as a function:

$$candidates : T \rightarrow 2^S$$

such that $\forall t \in T, \quad candidates(t) = \{s \in S \mid match(func_s(s), func_t(t))\}$, where *match* returns *true* if the functional description of the service semantically matches the functional requirements of the task (e.g., *match* can be aligned to a simple or complex semantic search in a UDDI registry).

Based on this, a graph of tasks can be *instantiated* to a graph of services by replacing each task t in the graph with a service $s \in candidates(t)$ providing the functionality required by the task. Formally, the possible *instances* of a task graph can be defined as a function *instances*, which maps a *dag* to a set of service graphs, each resulting from replacing the task nodes in the original graph with a particular combination of their candidate services:

$$instances : G \rightarrow 2^{G_s}$$

where G_s is the set of all *dag*s that can be formed from the services in S.

Example. Assume that the candidate services for achieving tasks E and F in Figure 3.1 are: $candidates(\mathrm{E}) = \{s_1, s_2\}$ and $candidates(\mathrm{F}) = \{s_3, s_4\}$. The task graph $g = (\{E, F\}, \{\langle E, F\rangle\})$ then has four possible instances:

$$instances(g) = \{(\{s_1, s_3\}, \{\langle s_1, s_3\rangle\}), (\{s_1, s_4\}, \{\langle s_1, s_4\rangle\}),$$
$$(\{s_2, s_3\}, \{\langle s_2, s_3\rangle\}), (\{s_2, s_4\}, \{\langle s_2, s_4\rangle\})\}$$

Now, the instantiation of a task's abstract plans can be defined as a function *comp*, which maps a task to a set of *actual plans* (instances of the task's abstract plans), each representing an alternative composite service for achieving the task:

$$comp : T \rightarrow 2^{G_s}$$

such that $\forall t \in T, \quad comp(t) = \{p_s = (V_s, E_s) \in G_s \mid \exists p = (V, E) \in plan(t), \quad p_s \in instances(p)\}$

3.6 Composite Service Quality Model

The value of a particular quality attribute for a composite service is an aggregation of the corresponding quality values for the component services, where the type of the aggregation function (e.g. summation, multiplication, min/max) depends on the quality attribute considered. Formally, the values of the quality attributes for a composite service can be defined as a function:

$$value_c : G_s \times NAME_a \rightarrow VALUE_a \cup \{\text{undf}\}$$

such that $\forall g_s = (V_s, E_s) \in G_s, \quad \forall a \in NAME_a,$

$$[value_c(g_s, a) = \text{undf}] \vee [value_c(g_s, a) = \boldsymbol{aggr}_{s \in V_s}(value_s(s, a)) \in dom_a(a)]$$

where ***aggr*** is some aggregation function that depends on the attribute considered. For example, aggregation functions for the quality attributes *execution time* (ex), *reliability* (rel), and *throughput* (thr) are defined in Figure 3.3. Here, function $mxpth(g_s)$ returns the path with the longest execution time in graph g_s, i.e. a path pth from a start node to a destination node in g_s with the maximum $\sum_{s \in nodes(pth)} (value_s(s, \text{ex}))$. Like before, $value_c(g_s, a) = \text{undf}$ indicates that the value is not defined (not applicable).

$$
\begin{aligned}
value_c(g_s, \mathrm{ex}) &= \sum_{s \in nodes(mxpth(g_s))} (value_s(s, \mathrm{ex})) \\
value_c(g_s, \mathrm{rel}) &= \prod_{s \in V_s} (value_s(s, \mathrm{rel})) \\
value_c(g_s, \mathrm{thr}) &= \min_{s \in V_s} (value_s(s, \mathrm{thr}))
\end{aligned}
$$

FIGURE 3.3: Examples of aggregation functions

Example. The execution time and reliability of composite service $(\{s_1, s_3\}, \{\langle s_1, s_3 \rangle\})$ are given as follows:

$$value_c((\{s_1, s_3\}, \{\langle s_1, s_3 \rangle\}), \mathrm{ex}) = value_s(s_1, \mathrm{ex}) + value_s(s_3, \mathrm{ex})$$

$$value_c((\{s_1, s_3\}, \{\langle s_1, s_3 \rangle\}), \mathrm{rel}) = value_s(s_1, \mathrm{rel}) * value_s(s_3, \mathrm{rel})$$

3.7 Request Model

In addition to specifying the goal task to be achieved, a user might impose constraints on the values of the quality attributes, and weight these attributes to reflect their relative importance to the user. Formally, a user request can be defined as a triple $(task_r, const_r, weight_r)$, where the components are as follows.

- $task_r \in T$ is the goal task to be accomplished.
- $const_r$ is an attribute constraint function, representing the QoS constraints imposed by the user for task $task_r$. It maps a quality attribute to an upper or lower user-defined bound for its value:

$$const_r : NAME_a \rightarrow VALUE_a \cup \{\mathrm{undf}\}$$

such that $\forall a \in NAME_a, \quad const_r(a) \in dom_a(a) \cup \{\text{undf}\}$. When $const_r(a) = \text{undf}$, there are no restrictions on the value of attribute a by the user. When $const_r(a) \neq \text{undf}$, $const_r(a)$ is either the minimum allowed value (if $dir_a(a) = \texttt{+}$) or the maximum allowed value (if $dir_a(a) = \texttt{-}$).

- $weight_r$ is an attribute weighting function, representing the user preferences regarding different quality attributes. It assigns to each quality attribute a user-defined weighting factor that reflects its relative importance to the user:

$$weight_r : NAME_a \rightarrow [0, 1]$$

such that $\sum_{a \in NAME_a} weight_r(a) = 1$.

Example. Suppose there is a request to achieve task A with the following preferences and constraints: the user is equally interested in maximising reliability (rel) and minimising execution time (ex), and the highest price (pr) the user is willing to pay is 50. This request can be represented as a triple $(task_r, const_r, weight_r)$:

- $task_r = \text{A}$
- $const_r(\text{pr}) = 50; \quad const_r(a) = \text{undf}, \ \forall a \neq \text{pr}$
- $weight_r(a) = 0.5, \ \forall a \in \{\text{rel}, \text{ex}\}; \quad weight_r(a) = 0, \ \text{otherwise}$

Based on the user-defined attribute weights, the quality values of a service (atomic or composite) can be aggregated into a single value, representing the *overall utility* of the service for the user. To simplify the definition of such utility, we first introduce the following functions.

- Function max_t returns the maximum value offered for a particular quality attribute by a task's candidate services:

$$max_t : T \times NAME_a \rightarrow VALUE_a$$

such that $\forall t \in T, \forall a \in NAME_a, \quad max_t(t,a) = \max_{s \in candidates(t)}(value_s(s,a))$

- Function min_t returns the minimum value offered for a particular quality attribute by a task's candidate services:

$$min_t : T \times NAME_a \rightarrow VALUE_a$$

such that $\forall t \in T, \forall a \in NAME_a, \quad min_t(t,a) = \min_{s \in candidates(t)}(value_s(s,a))$

- Function max_r returns the maximum value offered for a particular quality attribute by the requested task's actual plans:

$$max_r : NAME_a \rightarrow VALUE_a$$

such that:

$$\forall a \in NAME_a, \quad max_r(a) = \max_{p \in plan(task_r)}(\boldsymbol{aggr}_{t \in nodes(p)}(max_t(t,a)))$$

where ***aggr*** is the same aggregation function presented in the composite service quality model, and depends on the quality attribute considered (this function is increasing in the interval $[0, \infty)$).

- Function min_r returns the minimum value offered for a particular quality attribute by the requested task's actual plans:

$$min_r : NAME_a \rightarrow VALUE_a$$

such that:

$$\forall a \in NAME_a, \quad min_r(a) = \min_{p \in plan(task_r)} (\boldsymbol{aggr}_{t \in nodes(p)}(min_t(t,a)))$$

where ***aggr*** is the same aggregation function presented in the composite service quality model, and depends on the quality attribute considered (this function is increasing in the interval $[0, \infty)$).

- Function $scaled_s$ assigns to each atomic service its *scaled* value for a particular quality attribute:

$$scaled_s : T \times S \times NAME_a \rightarrow [0,1]$$

such that $\forall t \in T, \forall s \in candidates(t), \forall a \in NAME_a, \quad scaled_s(t,s,a)$ satisfies the following:

$$[(dir_a(a) = +) \Rightarrow (scaled_s(t,s,a) = \frac{value_s(s,a) - min_t(t,a)}{max_t(t,a) - min_t(t,a)})] \wedge$$
$$[(dir_a(a) = -) \Rightarrow (scaled_s(t,s,a) = \frac{max_t(t,a) - value_s(s,a)}{max_t(t,a) - min_t(t,a)})]$$

- Function $scaled_c$ assigns to each composite service its *scaled* value for a particular quality attribute:

$$scaled_c : G_s \times NAME_a \rightarrow [0,1]$$

such that $\forall p_s \in comp(task_r), \forall a \in NAME_a, \quad scaled_c(p_s,a)$ satisfies the following:

$$[(dir_a(a) = +) \Rightarrow (scaled_c(p_s,a) = \frac{value_c(p_s,a) - min_r(a)}{max_r(a) - min_r(a)})] \wedge$$
$$[(dir_a(a) = -) \Rightarrow (scaled_c(p_s,a) = \frac{max_r(a) - value_c(p_s,a)}{max_r(a) - min_r(a)})]$$

Now, the utility of an atomic service with respect to the user request can be defined as a function:

$$utility_s : T \times S \rightarrow [0, 1]$$

such that:

$$\forall t \in T, \forall s \in candidates(t), \quad utility_s(t, s) = \sum_{a \in NAME_a} (weight_r(a) * scaled_s(t, s, a))$$

Similarly, the utility of a composite service with respect to the user request can be defined as a function:

$$utility_c : G_s \rightarrow [0, 1]$$

such that:

$$\forall p_s \in comp(task_r), \quad utility_c(p_s) = \sum_{a \in NAME_a} (weight_r(a) * scaled_c(p_s, a))$$

Example. Consider two composite services p_s^1 and p_s^2, with scaled quality values for reliability and execution time being provided in Figure 3.4. The utility of each of these compositions, with respect to the previous example request, can thus be given as follows:

$$\begin{aligned} utility_c(p_s^1) &= (weight_r(\text{rel}) * scaled_c(p_s^1, \text{rel})) + (weight_r(\text{ex}) * scaled_c(p_s^1, \text{ex})) \\ &= (0.5 * 0.4) + (0.5 * 0.4) = 0.4 \\ utility_c(p_s^2) &= (weight_r(\text{rel}) * scaled_c(p_s^2, \text{rel})) + (weight_r(\text{ex}) * scaled_c(p_s^2, \text{ex})) \\ &= (0.5 * 0.2) + (0.5 * 0.8) = 0.5 \end{aligned}$$

	reliability	*execution time*
p_s^1	0.4	0.4
p_s^2	0.2	0.8

FIGURE 3.4: Scaled quality values of composite services p_s^1 and p_s^2

3.8 Service Selection Problem

The service selection problem involves finding the best combination of services to achieve the requested task, that both satisfies the imposed QoS constraints and maximises the overall utility with respect to the user-defined quality weights. Formally, the set of satisfactory composite services, SCS, meeting the user's quality constraints, can be defined as:

$$\begin{aligned} SCS = \{p_s \in comp(task_r) \mid \forall a \in NAME_a, \\ [(const_r(a) \neq \text{undf}) \wedge (dir_a(a) = \texttt{+}) \Rightarrow value_c(p_s, a) \geq const_r(a)] \quad \wedge \\ [(const_r(a) \neq \text{undf}) \wedge (dir_a(a) = \texttt{-}) \Rightarrow value_c(p_s, a) \leq const_r(a)]\} \end{aligned}$$

Now, the solution composite service, p_{sol}, for the user request is that satisfying the following: $p_{sol} \in SCS \ \wedge \ utility_c(p_{sol}) = \max_{p_s \in SCS}(utility_c(p_s))$.

3.9 Conclusion

In this chapter, we have presented and modelled formally the different components involved in a QoS-based service selection problem, which form the basis for our work. The model proposes a multi-granularity planning knowledge representation consisting of hierarchies of tasks, each with multiple alternative decompositions. The concept of task decomposition in our model is similar to some other work in the literature (e.g., task decomposition in HTN planning [111], and composite process decomposition in

the OWL-S process ontology [81]). The novelty of our formalism, however, is that it maps tasks at arbitrary levels of granularity to actual services, thus considering all possibilities to achieve the requested goal, unlike the existing approaches where only atomic tasks are mapped to actual services.

Task definition in our model is kept generic to be applicable to a wide range of domains. It may refer, for example, to an operation signature (in terms of input and output parameters), to a resource specification, or simply to a term of an ontology agreed within a community. Moreover, different mechanisms are possible for discovering suitable (candidate) services for each task: by consulting a central service repository storing service metadata (e.g., a semantic search over SAWSDL descriptions of web services advertised in a UDDI registry); or by calling for service proposals over the network (e.g., using the contract net protocol [112]). We make no assumptions in our model about any specific technology or service discovery and matching mechanism, and leave this to the application domain. Some concrete examples, however, are provided when presenting the case study evaluation (Chapter 8).

The QoS characteristics of services cover both generic attributes (e.g. execution time and cost) and domain-dependent ones (see Chapter 8 for examples). Again, we assume that suitable representation techniques are in place (or at least under ongoing research), without constraining the model to any particular implementation. Possible examples, out of many, could include Q-WSDL (QoS-enabled WSDL) [45] and the QoS Ontology by Maximilien and Singh [83]. Another example based on IEEE LOM metadata is provided in Chapter 8. Note that the model presented assumes that services are independent from each other regarding quality values. That is, the quality value offered by a service for a particular attribute in a composition is not affected by the other services comprising this composition. Since this may not always be the case, an

extension of the model addressing quality correlations among services is introduced in Chapter 7.

The request model assumes quantitative quality attributes spanning multiple tasks (i.e. at the goal task level). Such attributes, sometimes referred to as global (or end-to-end) attributes, represent the major challenge for the quality-aware service selection. This is because, as opposed to local (task-level) attributes, their satisfaction cannot be guaranteed by applying simple filtering per task prior to selection, but instead require more effective reasoning, taking all the tasks in a composition into account. That is, a large search space of the available service combinations need to be considered. Local attributes, however, can easily be added on top of our model.

In the next chapter, we suggest pruning techniques to reduce the search space, and introduce a selection algorithm to solve the quality-based service selection problem presented.

Chapter 4

Search Space Reduction Techniques for Efficient Service Selection

4.1 Introduction

Solving the service selection problem of Chapter 3 involves finding the best allocation of services to an abstract plan's tasks such that the global-level quality constraints are satisfied and the overall utility for the user is optimised. This is far from trivial with the increasing amount of the available services that offer similar functionalities, but differ in their qualities. Although many quality-based service selection algorithms have been proposed [145, 146, 13, 14, 31, 76, 143], these usually do not scale well when the number of candidate services becomes very large. In response, two types of pruning, namely task-based pruning and plan-based pruning, are introduced in this chapter, aiming to reduce the number of possible service combinations that need to be considered during

service selection; thus when applied *prior* to selection, they improve the computational cost.

The rest of the chapter is organised as follows. Section 4.2 presents our search space reduction techniques, while a quality-based service selection algorithm is introduced in Section 4.3. The proposed algorithm is evaluated analytically and empirically in Sections 4.4 and 4.5, respectively. Finally, Section 4.6 concludes the chapter.

4.2 Search Space Reduction

Since the number of possible actual plans (composite services) for achieving the goal task can be very large, service selection could suffer from a combinatorial explosion. This problem can be particularly acute in settings such as Grid computing, where each of thousands of nodes may act as a service for performing a particular processing job, but with widely varying speeds, robustness, storage capacity, etc. The selection complexity is affected by the number of tasks per abstract plan k, the number of candidate services per task n, and the number of alternative abstract plans p. For instance, given $k = 5$, $n = 1000$, and $p = 4$, the number of possible actual plans is $p \times n^k = 4 \times 1000^5$. Consequently, reducing this search space (the possible combinations of services) is essential to reduce the computational cost of the selection algorithm. To achieve search space reduction, this section presents two types of pruning, plan-based pruning and task-based pruning, which focus on reducing the number of alternative abstract plans p and the number of candidate services n, respectively.

4.2.1 Plan-based Pruning

As the composite service provider agent might have a number of possible alternative plans in its planning knowledge hierarchy to achieve the requested functionality, the goal

of plan-based pruning is to reduce this planning search space by eliminating those plans that will not lead to a satisfactory solution for the current request. More specifically, it identifies the abstract plans all of whose available instances (all possible combinations of services) are guaranteed to violate the issued quality constraints. In order to determine such unsatisfactory plans without performing the costly process of service selection first, we annotate each task in the planning knowledge hierarchy with information about its available candidate services. In particular, each task is associated with two items: the *number* of its candidate services; and its *ideal quality values*, which are obtained by assigning to each quality attribute the best value offered for this attribute among all the candidate services for the task. Formally, the ideal quality values for a task in the planning knowledge hierarchy can be defined as a function:

$$ideal_t : T \times NAME_a \rightarrow VALUE_a$$

such that $\forall t \in T,\ \forall a \in NAME_a$,

$$[(dir_a(a) = +) \Rightarrow (ideal_t(t, a) = max_t(t, a))] \wedge$$
$$[(dir_a(a) = -) \Rightarrow (ideal_t(t, a) = min_t(t, a))]$$

To illustrate the benefit of such information, consider the planning knowledge hierarchy of Figure 4.1, where the ideal quality values of each task correspond to price (pr) and execution time (ex) attributes. Here, there are five alternative plans for performing task A: *plan1*: A; *plan2*: B-C-D; *plan3*: G-C-D; *plan4*: B-C-E-F; and *plan5*: G-C-E-F. Now, suppose the user wants to achieve task A with the constraint that the cost should be less than 80. Since *plan1* has no available candidate services, it can be eliminated immediately from the planning search space. Based on the ideal quality values of tasks B, C, D, E, F and G, the best possible prices offered by *plan2*, *plan3*,

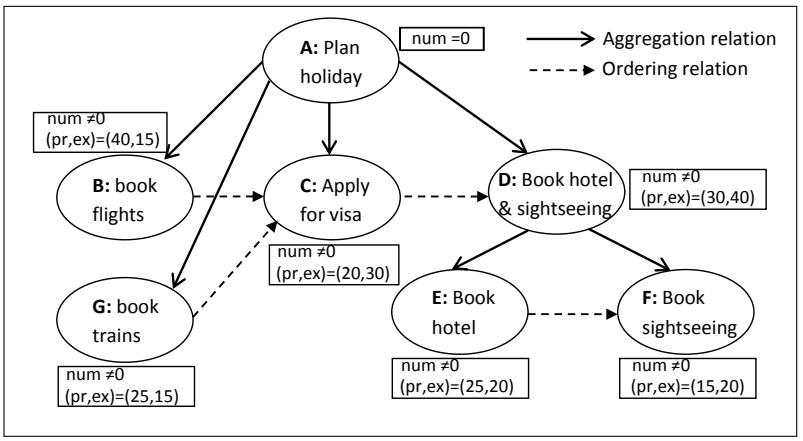

FIGURE 4.1: Annotating the tasks of the planning knowledge hierarchy

plan4, and *plan5* are 90, 75, 100, and 85, respectively. Being the only plan to satisfy the imposed price constraint, only *plan3* should be considered for composition while the other plans should be filtered out from the planning search space of the current request.

Formally, given a user request, the set of plans to be considered for composition according to plan-based pruning, $SPLAN$, can be defined as:

$$SPLAN = \{p = (V, E) \in plan(task_r) \mid \ (\forall t \in V,\ candidates(t) \neq \emptyset) \wedge (\forall a \in AR,$$
$$[(dir_a(a) = +) \Rightarrow \boldsymbol{aggr}_{t \in V}(ideal_t(t, a)) \geq const_r(a)] \wedge$$
$$[(dir_a(a) = -) \Rightarrow \boldsymbol{aggr}_{t \in V}(ideal_t(t, a)) \leq const_r(a)])\}$$

where $AR = \{a \in NAME_a \mid const_r(a) \neq \text{undf}\}$ is the set of the constrained quality attributes, and $\boldsymbol{aggr}$ is the same function presented in the composite service quality model (Section 3.6), which depends on the quality attribute considered.

Since the ideal quality values of tasks are independent of any particular user request, these can be precomputed for each task, and updated each time changes occur in the

environment, such as the addition of a new service, the deletion of an existing service, or changes in the quality values of a service.

4.2.2 Task-based Pruning

Task-based pruning aims to improve the efficiency of service selection by reducing the number of candidate services that need to be considered for each task and, as a result, reducing the number of possible combinations of services. For instance, in the case where there are 10 sub-tasks, each with 100 candidate services, removing one candidate service from any sub-task will eliminate 100^9 possible combinations from the problem search space. Two types of task-based pruning, namely *constraint-based pruning* and *domination-based pruning* are described below.

4.2.2.1 Constraint-based Pruning

Constraint-based pruning aims to eliminate the candidate services that result only in compositions violating the imposed quality constraints, and thus are not worth considering.

The common case is for the QoS values to deteriorate with composition. That is, the quality values offered by a composite service are worse than (or equal to) those of its components, e.g. price, reliability, execution time. In such a case, simple constraint-based pruning can be performed by removing all the candidate services that do not meet the constraints specified. Formally, $\forall t \in T$, candidate service $s \in candidates(t)$ should not be considered for composition according to constraint-based pruning **iff**:

$$\begin{aligned} \exists a \in AR, \quad & [(dir_a(a) = \texttt{+}) \Rightarrow (value_s(s,a) < const_r(a))] \wedge \\ & [(dir_a(a) = \texttt{-}) \Rightarrow (value_s(s,a) > const_r(a))] \end{aligned}$$

This pruning, however, could eliminate satisfactory solutions from the search space when the composite service provides better quality values than its comprising services. Such a case could occur, for example, when the user is interested in maximising the overall duration of composition due to domain-specific characteristics (see Chapter 8 for more details). Consequently, the pruning should be improved by taking into consideration other tasks, as follows: $\forall t \in T$, candidate service $s \in candidates(t)$ should not be considered for composition according to constraint-based pruning **iff**:

$$\forall p = (V, E) \in plan(task_r) \text{ s.t. } t \in V, \quad \exists a \in AR,$$
$$[(dir_a(a) = +) \Rightarrow (\boldsymbol{aggr}_{v \in V}(vl(v, a)) < const_r(a))] \wedge$$
$$[(dir_a(a) = -) \Rightarrow (\boldsymbol{aggr}_{v \in V}(vl(v, a)) > const_r(a))]$$

where $vl(v, a) = value_s(s, a)$ if $v = t$; and $vl(v, a) = ideal_t(v, a)$, otherwise.

4.2.2.2 Domination-based Pruning

Domination-based pruning reduces the search space by filtering out uninteresting services from the set of candidates services for each task. Such uninteresting services are *dominated* by another candidate service for the same task, with a service s_1 being dominated by another service s_2 **iff** s_1 is worse than s_2 with respect to all quality attributes. For example, a flight booking service with price *pr:20* and execution time *ex:10* dominates (is better than) such a service with *pr:30* and *ex:20*. Formally,

$\forall t \in T,\ \forall s_1, s_2 \in candidates(t),\quad s_2$ dominates (dm) $s_1 \quad \Leftrightarrow$

$$\begin{aligned}(\forall a \in NAME_a,\ & [(dir_a(a) = +) \Rightarrow (value_s(s_1, a) \leq value_s(s_2, a))] \wedge \\ & [(dir_a(a) = -) \Rightarrow (value_s(s_1, a) \geq value_s(s_2, a))]) \wedge \\ (\exists a \in NAME_a,\ & [(dir_a(a) = +) \Rightarrow (value_s(s_1, a) < value_s(s_2, a))] \wedge \\ & [(dir_a(a) = -) \Rightarrow (value_s(s_1, a) > value_s(s_2, a))])\end{aligned}$$

Dominated services are not potential candidates for the optimal solution. To illustrate, consider two composite services:

$$g_s = (V_s = \{s_1, s_2, ..., s_i, ..., s_n\}, E_s); \quad g'_s = (V'_s = \{s_1, s_2, ..., s'_i, ..., s_n\}, E'_s)$$

where service s_i is dominated by s'_i. Since the aggregation functions in the composite service quality model are increasing in the interval $[0, \infty)$, the following is satisfied:

$$\begin{aligned}\forall a \in NAME_a,\ & [(dir_a(a) = +) \Rightarrow (value_c(g'_s, a) \geq value_c(g_s, a))] \wedge \\ & [(dir_a(a) = -) \Rightarrow (value_c(g'_s, a) \leq value_c(g_s, a))]\end{aligned}$$

In other words, the quality values of composite service g'_s are either equal to or better than those of composite service g_s, for all quality attributes. Thus, given that g_s satisfies the quality constraints, g'_s will also meet the same constraints, but with better utility. Hence, keeping only non-dominated services as candidates for each task will still guarantee finding the optimal solution for the selection problem (when such a solution exists) while reducing the computational cost.

Note that since the best (non-dominated) services for each task are independent of any particular user request, they need not be determined at request time. Instead, these

TABLE 4.1: Request-independent non-dominated services

	price	*execution time*	*availability*	*reliability*
s_1	10	20	40	10
s_2	30	40	10	40
s_3	5	25	30	20
s_4	6	27	20	15

services can be pre-computed for each task in the planning knowledge hierarchy, with this knowledge being continuously modified in response to changes in the environment.

Although this request-independent pruning (*ri*-pruning) keeps, for each task, only the best candidate services with respect to all quality attributes, some services might still be dominated by others when considering a particular user request, and thus are not candidates for this request's optimal solution. For instance, suppose the candidate services for a particular task after such domination-based pruning are given in Table 4.1, where each service is annotated with four QoS attributes (for simplicity, the values of all the attributes are normalised so that lower values mean better quality). Now, suppose the user is interested in a composite service that meets a constraint $const_r(\text{ex})$ on execution time and has the minimum price, regardless of its availability and reliability. Here, service s_4 is dominated by service s_3 since the latter has lower price and shorter execution time. Consequently, for each composite service satisfying $const_r(\text{ex})$ and containing s_4 as a member, another satisfactory composite service with better utility (lower price in this example) can be found by replacing s_4 with s_3. Similarly, s_1 dominates s_2. Hence only s_1 and s_3 should be considered for composition.

Based on this example, further search space reduction can be achieved by removing, from the set of *request-independent* non-dominated candidate services for each task, those candidates that are dominated by another candidate *according to the current request*. A service s_1 is dominated by another service s_2 with respect to a certain request **iff** s_1 is worse than s_2 regarding all the *constrained* quality attributes and the *utility* value. This additional *request-dependent* domination pruning (*rd*-pruning)

results in a considerable reduction in the number of candidate services for composition without affecting the ability to find the optimal solution. Formally, $\forall t \in T,\ \forall s_1, s_2 \in candidates(t),\ \ s_2$ request-based dominates (r-dm) $s_1 \ \Leftrightarrow$

$$\begin{aligned}(\forall a \in AR,\ \ & [(dir_a(a) = +) \Rightarrow (value_s(s_1, a) \leq value_s(s_2, a))] \wedge \\ & [(dir_a(a) = -) \Rightarrow (value_s(s_1, a) \geq value_s(s_2, a))]) \wedge \\ & (utility_s(t, s_1) \leq utility_s(t, s_2)) \wedge \\ ((\exists a \in AR,\ \ & [(dir_a(a) = +) \Rightarrow (value_s(s_1, a) < value_s(s_2, a))] \wedge \\ & [(dir_a(a) = -) \Rightarrow (value_s(s_1, a) > value_s(s_2, a))]) \vee \\ & (utility_s(t, s_1) < utility_s(t, s_2)))\end{aligned}$$

According to this request-based dominance relation, we define the following functions.

- Function *rcandidates* maps a task to its request-based non-dominated candidate services:

  $$rcandidates : T \rightarrow 2^S$$

  such that $\forall t \in T$,

  $$rcandidates(t) = \{s_i \in candidates(t) \mid \forall s_j \neq s_i \in candidates(t),\ \ \neg(s_j \text{ r-dm } s_i)\}$$

- Function $dominated_s$ maps a service $s \in candidates(t)$, to the services from $candidates(t)$ request-based dominated by this service:

  $$dominated_s : S \rightarrow 2^S$$

such that $\forall t \in T,\ \forall s \in candidates(t)$,

$$dominated_s(s) = \{s_i \in candidates(t) \mid s \text{ r-dm } s_i\}$$

- Function $dominating_s$ maps a service $s \in candidates(t)$, to the services from $rcandidates(t)$ request-based dominating this service:

$$dominating_s : S \rightarrow 2^S$$

such that $\forall t \in T,\ \forall s \in candidates(t)$,

$$dominating_s(s) = \{s_i \in rcandidates(t) \mid s_i \text{ r-dm } s\}$$

4.3 Service Selection

Several methods have been proposed to tackle the global optimal service selection problem. Here, we model this as a multi-constrained optimal path selection problem in a directed graph (sometimes called *multi-constrained QoS routing*), which involves finding a path, from the start node to the destination node in a network, that satisfies a set of path-level QoS constraints while being optimal with respect to some cost function. Modelling the service selection problem in this way allows us to cope with all alternative abstract plans at once, without applying the selection method on each plan separately.

Several algorithms have been proposed as an attempt to solve the QoS routing problem. Here we adopt the multi-constrained Bellman-Ford algorithm (an extension of the original Bellman-Ford shortest path algorithm) [144], which we first introduce, and then show how it can be updated to solve our service selection problem.

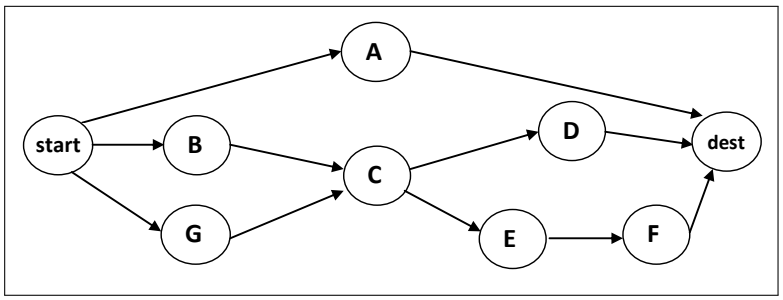

FIGURE 4.2: Plan paths graph

4.3.1 Multi-constrained Bellman-Ford Algorithm

Given a set of path-level QoS constraints, the multi-constrained Bellman-Ford algorithm stores in each node v of the search graph, the optimal paths discovered so far from the start node to v, with a path being considered optimal if no other path in the graph (from the same source to the same destination) is better for all the constrained quality attributes. Note that, since the optimisation is performed with respect to multiple objectives (the constrained attributes), several optimal paths could be recorded at each node, as opposed to the original Bellman-Ford algorithm where each node only stores one path (due to optimising a single objective).

The optimal paths from the start node to all other nodes are generated by traversing the graph edges, and for each edge (u, v), comparing the optimal paths already recorded in v with those obtained by joining the optimal paths in u with edge (u, v). As a result of processing all graph edges $|V| - 1$ times ($|V|$ is the number of nodes in the graph), the destination node contains all optimal paths from the start node to the destination, of which, those that satisfy the QoS requirements are considered solution paths.

TABLE 4.2: Valid predecessors for nodes of Figure 4.2 in the running example (S = start node)

Node	A	B	C	D	E	F	G	*dest*
validpred	$\emptyset$	S	SB SG	SBC SGC	SGC	$SGCE$	S	$SBCD$ $SGCD$ $SGCEF$

4.3.2 Selection Algorithm

The first step in our service selection algorithm is to generate the plan paths graph (V_{PK}, E_{PK}), where each path corresponds to an alternative abstract plan for achieving the requested task. Note that all abstract plans are assumed to be sequential (other structures can be transformed to the sequential structure using existing techniques [33]). Figure 4.2 provides the plan paths graph for the planning knowledge of Figure 4.1 (in which the start and end nodes are additional nodes that are ignored when instantiating the graph paths). Based on the idea of the Bellman-Ford algorithm, each node v in the plan paths graph stores the optimal instances (optimal composite services) for each path discovered so far from the start node to v. For example, node C in Figure 4.2 records the optimal instances of paths BC and GC. In order to maximise utility, the concept of optimal paths in the original Bellman-Ford algorithm is updated so that an instance of a path is considered optimal if no other possible instance of the same path has both better values for all the constrained attributes and better utility. Moreover, to reduce the number of optimal instances, only those satisfying the quality constraints are maintained in each node. After traversing all graph nodes in topological order, the solution is the optimal composite service with the best utility at the destination node.

In order to account for plan-based pruning, we maintain in each node v of the plan paths graph the set of its valid predecessors $validpred(v)$, which is the set of paths from the start node to v that, when combined with at least one path from v to the end node, will lead to a satisfactory abstract plan with respect to the plan-based pruning.

In other words, given a path $p_v + v$ from the start node to v, path p_v is considered a valid predecessor of node v *iff* there exists at least one path p_i from v to the destination node, such that $p_v + p_i \in SPLAN$. Based on this, the processing can be updated so that each node only stores the optimal instances for its valid predecessors. Formally, the optimal instances *oinstances*, of a node for a particular valid predecessor can be defined as:

$$oinstances : T \times G \rightarrow 2^{G_s}$$

such that $\forall v \in T,\ \forall p_v \in validpred(v),\quad oinstances(v, p_v) =$

$$\{ins_i \in instances(p_v + v) \mid \forall ins_j \in instances(p_v + v),\quad \neg(ins_j \text{ r-dm } ins_i)\}$$

The request-based dominance relation, r-dm, defined on instances (composite services) is similar to that of atomic services (see Section 4.2.2.2), replacing $value_s$ and $utility_s$ with $value_c$ and $utility_c$, respectively.

Example. Given that $SPLAN = \{BCD, GCD, GCEF\}$, the valid predecessors of the nodes in Figure 4.1 are provided in Table 4.2. Hence, when processing edge (C, E), all the optimal composite services stored at node C that are instances of path BC should be ignored (SBC is not a valid predecessor of node E), and only those that are instances of path GC should be considered for joining with node E's candidate services.

This service selection algorithm is provided in Algorithm 1. Function $pcandidates(u)$ at line 3 of the *process-edge* procedure returns the candidate services of node u that survive task-based pruning ($pcandidates(u)$=$candidates(u)$ when no task pruning is performed).

Algorithm 1 Selection-Algorithm

1: generate plan paths graph $g_{PK} = (V_{PK}, E_{PK})$
2: assign to each node $v \in V_{PK} \setminus \{v_{start}\}$ its valid predecessors $validpred(v)$
3: sort the nodes in V_{PK} topologically
4: store the empty instance at v_{start} and empty service at v_{dest}
5: **for** each $v \in V_{PK} \setminus \{v_{dest}\}$ taken in topological order **do**
6: **if** $v = v_{start}$ **or** $validpred(v) \neq \emptyset$ **then**
7: **for** each $(v, u) \in E_{PK}$ **do**
8: process-edge(v,u)
9: Solution ins_{sol} is the optimal instance at v_{dest} with the highest utility $utility_c$

Procedure 2 process-edge(v,u)

1: **for** each $p_u \in validpred(u)$ **do**
2: **if** $v = enode(p_u)$ **then**
3: **for** each $s \in pcandidates(u)$ **do**
4: **for** each optimal instance $ins_v \in oinstances(v, p_u - v)$ **do**
5: **if** instance $ins_v + s$ is satisfactory **then**
6: check-instance-optimality($ins_v + s$,u,p_u)

Procedure 3 check-instance-optimality(ins,u,p_u)

1: optml $\leftarrow$ 1
2: **for** each optimal instance $ins_u \in oinstances(u, p_u)$ **do**
3: **if** ins_u r-dm ins **then**
4: optml $\leftarrow$ 0
5: **break**
6: **else if** ins r-dm ins_u **then**
7: $oinstances(u, p_u) \leftarrow oinstances(u, p_u) \setminus \{ins_u\}$
8: **if** optml=1 **then**
9: $oinstances(u, p_u) \leftarrow oinstances(u, p_u) \cup \{ins\}$

4.4 Analytical Study

The goal of this section is to analyse the time complexity of the proposed selection algorithm. For simplicity, the analysis is first provided for the specific case of one abstract plan, and then generalised to handle the planning knowledge hierarchy (i.e. multiple abstract plans).

4.4.1 One Abstract Plan

Consider a sequential abstract plan comprised of k tasks, $v_1v_2...v_k$, each with n available candidate services. The time complexity of the proposed selection algorithm, $\tau(alg_s)$, can be computed in terms of the pre-selection pruning time, $\tau(prn)$, and selection time, $\tau(sel)$, as follows:

$$\tau(alg_s) = \tau(prn) + \tau(sel) \tag{4.1}$$

To prune uninteresting candidate services for task node v_i, each candidate of this task may need to be compared with all the remaining candidates, i.e. there are $n \times (n-1)$ comparisons in the worst case. Hence, $\tau(prn)$ is $O(k \times n^2)$.

To select the optimal solution (the best instance of path $v_1v_2...v_k$), each node $v_{i>1}$ records the optimal instances of path $v_1v_2...v_i$, denoted $oinstances(v_i)$. Hence:

$$\tau(sel) = \sum_{i=2}^{k} \tau(oinstances(v_i)) \tag{4.2}$$

The time required to calculate $oinstances(v_i)$, $\tau(oinstances(v_i))$, depends on the sizes of $oinstances(v_{i-1})$ and $pcandidates(v_i)$, which can be defined in terms of the following pruning rates.

- $spr_i \in [0,1]$, the percentage of candidate services pruned per task node v_i prior to selection. That is, $|pcandidates(v_i)| = n \times sr_i$, where $sr_i = 1 - spr_i$. Note that $spr_i = 0$ if no task-based pruning is performed.
- $ipr_i \in [0,1]$, the percentage of candidate instances pruned per path $v_1v_2...v_i$ when computing the optimal instances at node v_i. That is:

$$|oinstances(v_i)| = |oinstances(v_{i-1})| \times |pcandidates(v_i)| \times ir_i \tag{4.3}$$

(where $ir_i = 1 - ipr_i$)

Since $|oinstances(v_1)| = |pcandidates(v_1)| = n \times sr_1$, $|oinstances(v_i)|$ becomes as follows:

$$|oinstances(v_i)| = n^i \times \prod_{m=1}^{i} sr_m \times \prod_{m=2}^{i} ir_m \tag{4.4}$$

Based on this, and since each candidate instance at node v_i may need to be compared with all the remaining instances in the worst case, we conclude that:

$$\begin{aligned} \tau(oinstances(v_i)) \text{ is } & O((|oinstances(v_{i-1})| \times |pcandidates(v_i)|)^2) \\ \text{is } & O((n^{i-1} \times \prod_{m=1}^{i-1} sr_m \times \prod_{m=2}^{i-1} ir_m \times n \times sr_i)^2) \\ \text{is } & O(n^{2i} \times \prod_{m=1}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2) \end{aligned} \tag{4.5}$$

Assume for simplicity that $\forall i,\ sr_i = ir_i = r$. In this case, $\tau(oinstances(v_i))$ is $O(n^{2i} \times r^{4i-4})$, and selection complexity $\tau(sel)$ is defined by the following two lemmas.

Lemma 4.1. *When $r^2 > \frac{1}{n}$, our algorithm achieves a selection complexity $\tau(sel)$ of $O(n^\alpha)$, such that $\alpha = 2k + \log_n(r^{4k-4})$.*

Proof. Since $\tau(oinstances(v_i))$ is $O(n^{2i} \times r^{4i-4})$, the following holds if $r^2 > \frac{1}{n}$:

$$\forall i \in [2, k-1],\ \tau(oinstances(v_i)) < \tau(oinstances(v_{i+1}))$$

From Equation 4.2, and assuming k is a small number, i.e. $k << n$, we can conclude that $\tau(sel)$ is $O(\tau(oinstances(v_k)))$. That is, $\tau(sel)$ is $O(n^\alpha)$, with:

$$\begin{aligned} & n^\alpha = n^{2k} \times r^{4k-4} \quad \text{(see Equation 4.5)} \\ \Rightarrow\ & n^{\alpha-2k} = r^{4k-4} \\ \Rightarrow\ & \alpha = 2k + \log_n(r^{4k-4}) \end{aligned}$$

□

Lemma 4.2. *When $r^2 \leq \frac{1}{n}$, our algorithm achieves a selection complexity $\tau(sel)$ of $O(n^\alpha)$, such that $\alpha = 4 + \log_n(r^4)$.*

Proof. Since $\tau(oinstances(v_i))$ is $O(n^{2i} \times r^{4i-4})$, the following holds if $r^2 \leq \frac{1}{n}$:

$$\forall i \in [2, k-1],\ \tau(oinstances(v_i)) \geq \tau(oinstances(v_{i+1}))$$

From Equation 4.2, and assuming k is a small number, i.e. $k << n$, we can conclude that $\tau(sel)$ is $O(\tau(oinstances(v_2)))$. That is, $\tau(sel)$ is $O(n^\alpha)$, with:

$$\begin{aligned} & n^\alpha = n^4 \times r^4 \quad \text{(see Equation 4.5)} \\ \Rightarrow\ & n^{\alpha-4} = r^4 \\ \Rightarrow\ & \alpha = 4 + \log_n(r^4) \end{aligned}$$

□

Example. Suppose that $n = 100$ and $r = 10\%$ (i.e. the pruning rate is 90%). From Lemma 4.2, the selection process is of time complexity $O(n^2)$, since $\alpha = 4 + \log_{100}(0.0001) = 2$.

Based on the above, the overall time complexity of our selection algorithm $\tau(alg_s)$ is $O(n^{max(2,\alpha)})$ (see Equation 4.1).

4.4.2 Multiple Abstract Plans

Given a planning knowledge hierarchy with l levels, d decomposition graphs per parent task, and k sub-tasks per decomposition graph, the number of alternative abstract plans for the goal task (root), $plnnum(l, d, k)$, is:

$$plnnum(l, d, k) = 1 + d \times plnnum(l - 1, d, k)^k \tag{4.6}$$

with $\forall k, d, \quad plnnum(1, d, k) = 1$.

Based on this, Lemmas 4.1 and 4.2 can be generalised as follows.

Lemma 4.3. *When $r^2 > \frac{1}{n}$, our algorithm achieves a selection complexity $\tau(sel)$ of $O(n^\alpha)$, such that:*

$$\alpha = 2 \times tmx + \log_n(pr \times plnnum(l, d, k) \times r^{4 \times tmx - 4})$$

where pr is the percentage of abstract plans surviving plan-based pruning; and $tmx \leq k^{l-1}$ is the maximum number of tasks per abstract plan (among the surviving plans).

Proof. The proof can be easily derived similarly to Lemma 4.1, given that $\tau(sel)$ is $O(pr \times plnnum(l, d, k) \times \tau(oinstances(v_{tmx})))$. □

Lemma 4.4. *When $r^2 \leq \frac{1}{n}$, our algorithm achieves a selection complexity $\tau(sel)$ of $O(n^\alpha)$, such that:*

$$\alpha = 4 + \log_n(pr \times plnnum(l, d, k) \times r^4)$$

where pr is the percentage of abstract plans surviving plan-based pruning.

Proof. The proof can be easily derived similarly to Lemma 4.2, given that $\tau(sel)$ is $O(pr \times plnnum(l, d, k) \times \tau(oinstances(v_2)))$. □

4.5 Empirical Study

This section evaluates the effectiveness of the pruning techniques presented, focusing mainly on assessing the gain in performance, in terms of computation time. For this purpose, the candidate services of each task are generated randomly (each candidate service is assumed to have 5 quality attributes). Request quality constraints and quality weights are also set to random values. For simplicity, each parent task in the planning knowledge hierarchy is assumed to have one decomposition graph, and all quality attributes are assumed to be additional (***aggr*** is the sum function), with their values worsening with composition. The other, less common, case where the quality values could improve with composition is evaluated in Chapter 8. All experiments were conducted on a PC with Intel Core 2 Quad 3GHz CPU and 3.23GB RAM.

To evaluate the gain in performance achieved by applying domination-based pruning prior to service selection, the computation time of the selection algorithm (in relation to both the number of candidate services per task, and the number of the constrained quality attributes) was compared in three cases: no task pruning (the basic algorithm), request-independent domination pruning alone, and both request-independent and request-dependent domination pruning. Here, the number of candidate services per task was varied between 100 and 1000, while the number of levels in the hierarchy and the number of child tasks for each parent were fixed at 3 and 4, respectively (i.e. the number of tasks in each abstract plan $\leq$ 16). The execution time of each

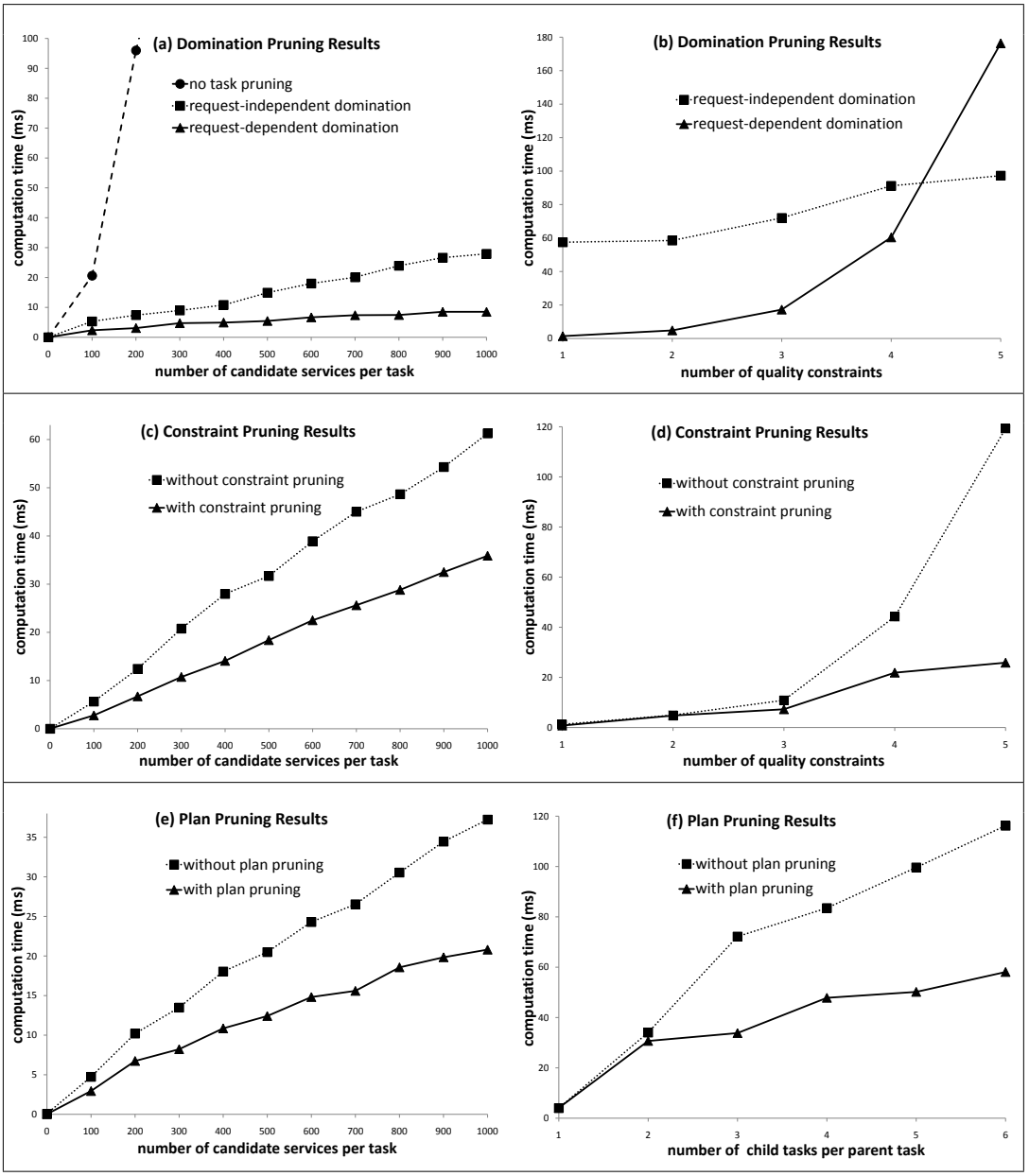

FIGURE 4.3: Evaluating the search space reduction techniques

algorithm was averaged over 20 different graph instances, and 20 different randomly-generated requests. The results, provided in Figure 4.3(a), indicate that both domination types significantly outperform the basic algorithm whose execution time increases dramatically with the increasing number of candidate services. Moreover, we observe that request-dependent pruning achieves better performance than request-independent pruning, especially as the number of candidate services grows.

To further compare the performance of the two domination approaches with respect to the number of quality constraints, the number of candidate services per task was fixed at 1000, while the number of the constrained attributes was varied between 1 and 5 (5 is the total number of quality attributes). As before, the reported computation times are averaged (over 20 graph instances and 10 requests). As shown in Figure 4.3(b), both algorithms require more time when the problem becomes more constrained because, as the number of constraints increases, so does the number of parameters with respect to which the optimal instances are computed, resulting in more optimal instances being stored at each graph node.

Although the request-dependent pruning approach is generally more efficient than the request-independent pruning one, the latter becomes more efficient than the former when the number of quality constraints is maximal (5 in our case). This behaviour is expected due to the fact that, when the number of quality constraints is equal to the total number of quality attributes, no service will be request-based dominated by another service after applying request-independent pruning to each task. Hence, request-dependent pruning in this case only results in adding unnecessary overhead to overall performance (this type of pruning is performed at request time), without achieving any further reduction in the service search space.

To study the improvement in execution time as a result of constraint-based pruning,

the time of the selection algorithm was compared in two cases: domination-based pruning alone; and both domination-based pruning and constraint-based pruning. Again, the effects of the numbers of candidates and constraints were considered, with the experimental settings being the same as previously.

The results in Figure 4.3(c) show that constraint pruning in combination with domination pruning performs better than domination pruning alone. Clearly, the gain in performance increases with the increasing number of quality constraints, as shown in Figure 4.3(d) because, in highly constrained problems, more candidate services are likely to violate the quality constraints, and thus are pruned prior to applying the selection algorithm, resulting in computation time reduction.

Finally, to assess the effectiveness of plan-based pruning, task-based pruning (domination pruning and constraint pruning) alone was compared with the combination of task-based pruning and plan-based pruning. Figures 4.3(e) and 4.3(f) show the running time of the algorithms, varying the number of candidate services per task and the number of child tasks per parent task, respectively. As expected, the inclusion of plan-based pruning results in an improvement in performance: the computation time reduction increases as the problem becomes more complex (either as a result of increasing the number of candidate services, or increasing the number of alternative abstract plans).

4.6 Conclusion

In this chapter, we have presented an efficient approach to solve the QoS-based service selection problem of Chapter 3. Specifically, to reduce the selection search space, two types of pruning, namely plan-based pruning and task-based pruning, were suggested.

The former filters out the abstract plans whose available instances are collectively unsatisfactory, thus decreasing the number of plans to be considered for selection. The later, on the other hand, focuses on reducing the number of services for a task by eliminating those services guaranteed not to be part of the solution composition, due either to being unsatisfactory or to not representing potential candidates for optimality, determined in terms of the dominance relation. As opposed to some existing heuristic attempts to reduce the number of services (e.g. by Qi et al. [101] and Trummer et al. [127]), our task-based pruning does not affect the ability to find an optimal solution. Moreover, although the notion of dominance is not new (e.g. used by Alrifai et al. [5] and Hiratsuka et al. [59]), we presented a *request-based* dominance, where the domination among services is identified with respect to user's request, thus resulting in more uninteresting services being pruned for each task.

To evaluate the effectiveness of the suggested pruning techniques, a service selection algorithm utilising the optimal paths concept of the multi-constrained Bellman-Ford algorithm was introduced. Results show that a significant gain in performance is achieved through applying our pruning techniques prior to selection. Note that these techniques are not constrained to the selection algorithm proposed, and can be seen as a preprocessing step to any selection algorithm to reduce its complexity.

The selection algorithm currently assumes a sequential flow for each abstract plan. Many real-world applications, however, are not necessarily structured according to a strict sequential order, but rather encounter some parallelism or even loops. Extending the algorithm to accommodate such structures (e.g. parallel and loop structures) will not be addressed in this thesis. Furthermore, the service landscape is assumed to be static during the selection process. Yet, services operate in highly dynamic environments, resulting in the possibility of service changes occurring during selection. If ignored, these changes could lead to selecting an unsatisfactory, non-executable,

or non-optimal solution. Consequently, in the next chapter, we eliminate the static world assumption at selection time and present a reactive selection algorithm capable of adapting to service changes while performing selection.

Chapter 5

Addressing Changes during Service Selection

5.1 Introduction

Service-based systems exhibit high degrees of dynamism and uncertainty due to the dependency on remote services, which are typically owned by autonomous and self-interested participants. Specifically, existing providers may choose not to fulfil their promises, to upgrade or degrade their quality offerings (e.g., to increase competitiveness), or to disconnect from the system at any time, while new providers might join instead. Even with long-standing and cooperative providers, availability and quality estimates of services could still frequently change over time due to other factors. For instance, a service's response time could be significantly affected by the provider load and network traffic at that particular moment. Similarly, a service might suddenly become unavailable due to network or hardware failure.

Being unpredictable, it is likely that such a change in the service landscape may be encountered during selection (i.e., when tasks have been partially, or even fully, processed for service allocation, but before the start of execution). Since service effects have not yet taken place, recovery at this stage involves no (serious) complications, but simply modifying the assignments of services to tasks. However, if the change is not handled in time and is instead left to be detected during execution when the faulty or quality violating service is invoked, as in most existing composition approaches [30, 14, 12, 25, 78, 92, 29], it can result in undesired situations. This includes execution disruption due to time-consuming replanning, and possibly no longer optimal or, in the worst case, an unsatisfactory solution given that services have already executed, which may require operation compensations. This is in addition to losing the chance of exploiting emerging better opportunities, if neglected during selection.

In response, this chapter builds on the selection algorithm of Chapter 4, but significantly improves it by presenting a novel reactive version, capable of adapting efficiently to service changes during *selection*, with a minimal number of modifications, thus ensuring, when execution starts, that the selected composite service is executable, satisfactory, and optimal according to the current environment state.

The rest of the chapter is organised as follows. Section 5.2 presents a motivating example, followed by our reactive selection algorithm in Section 5.3. The proposed algorithm is evaluated analytically and empirically in Sections 5.4 and 5.5, respectively. Finally, Section 5.6 concludes the chapter.

5.2 Motivating Example

To illustrate the need for reactive service selection that can handle changes to services during the selection process, consider an example in which the user has issued a request

to achieve task A, and is interested in minimising the price (pr) while satisfying the following constraint on execution time: ex $\leq$ 100. The plan paths graph for the requested task, and the request-based non-dominated candidate services of the involved sub-tasks are depicted in Figure 4.2 and Table 5.1, respectively. According to plan-based pruning, the set of plans to be considered for accomplishing task A, $SPLAN = \{BCEF\}$ (plans A, BCD, GCD, and $GCEF$ are excluded from this set since tasks A and D do not have any available services, and all the available instances of plan $GCEF$ violate the imposed execution time constraint). Given set $SPLAN$, the valid predecessors of the nodes are as provided in Table 5.1.

TABLE 5.1: Request-based non-dominated services and valid predecessors for the nodes of Figure 4.2 (S = start node)

Node	A	B	C	D	E	F	G
$rcandidates$ (ex,pr)	$\emptyset$	$s_{B1}(20,30)$ $s_{B2}(30,12)$	$s_{C1}(15,50)$ $s_{C2}(30,30)$	$\emptyset$	$s_{E1}(27,5)$ $s_{E2}(15,20)$	$s_{F1}(30,10)$ $s_{F2}(20,40)$	$s_{G1}(86,5)$
$validpred$	$\emptyset$	S	SB	$\emptyset$	SBC	$SBCE$	$\emptyset$

Now, suppose the selection algorithm visits the nodes in the following topological order: *start*, A, G, B, C, E, F and D. Visiting the start node *start* involves processing its outgoing edge $(start, B)$, with edges $(start, A)$ and $(start, G)$ being ignored since $validpred(A) = validpred(G) = \emptyset$. This results in storing the request-based non-dominated services of node B as its optimal instances (see Table 5.2). Nodes A and G do not have any valid predecessors, and thus are ignored in the traversal. Next, node B is visited, and edge (B, C) is processed by combining the optimal instances at node B with node C's candidate services, resulting in four optimal instances of path BC, as shown in Table 5.2. After traversing node C, six optimal instances of path BCE are maintained in node E (Table 5.2). Being dominated by other instances (indicated by symbol **d**), instances $s_{B1}s_{C1}s_{E1}$ and $s_{B1}s_{C2}s_{E1}$ are not considered optimal, and thus are not recorded at node E. Similarly, as provided in Table 5.2, visiting node E

TABLE 5.2: Optimal instances estimation for nodes B, C, E and F (S = start node)

Node	B	C	E	F	
validpred	S	SB	SBC	$SBCE$	
oinstances (ex,pr)	$s_{B1}(20,30)$ $s_{B2}(30,12)$	$s_{B1}s_{C1}(35,80)$ $s_{B1}s_{C2}(50,60)$ $s_{B2}s_{C1}(45,62)$ $s_{B2}s_{C2}(60,42)$	$s_{B1}s_{C1}s_{E1}(62,85)$**(d)** $s_{B1}s_{C1}s_{E2}(50,100)$ $s_{B1}s_{C2}s_{E1}(77,65)$**(d)** $s_{B1}s_{C2}s_{E2}(65,80)$ $s_{B2}s_{C1}s_{E1}(72,67)$ $s_{B2}s_{C1}s_{E2}(60,82)$ $s_{B2}s_{C2}s_{E1}(87,47)$ $s_{B2}s_{C2}s_{E2}(75,62)$	$s_{B1}s_{C1}s_{E2}s_{F1}(80,110)$ $s_{B1}s_{C1}s_{E2}s_{F2}(70,140)$ $s_{B1}s_{C2}s_{E2}s_{F1}(95,90)$ $s_{B1}s_{C2}s_{E2}s_{F2}(85,120)$**(d)** $s_{B2}s_{C1}s_{E1}s_{F1}(102,77)$**(ns)** $s_{B2}s_{C1}s_{E1}s_{F2}(92,107)$**(d)**	$s_{B2}s_{C1}s_{E2}s_{F1}(90,92)$ $s_{B2}s_{C1}s_{E2}s_{F2}(80,122)$**(d)** $s_{B2}s_{C2}s_{E1}s_{F1}(117,57)$**(ns)** $s_{B2}s_{C2}s_{E1}s_{F2}(107,87)$**(ns)** $s_{B2}s_{C2}s_{E2}s_{F1}(105,72)$**(ns)** $s_{B2}s_{C2}s_{E2}s_{F2}(95,102)$**(d)**

TABLE 5.3: Optimal instances reestimation in response to change 1 (S = start node)

Node	E	F	
validpred	SBC	$SBCE$	
oinstances (ex,pr)	$s_{B1}s_{C1}s_{E1}(62,85)$ $s_{B1}s_{C2}s_{E1}(77,65)$ $s_{B2}s_{C1}s_{E1}(72,67)$ $s_{B2}s_{C2}s_{E1}(87,47)$	$s_{B1}s_{C1}s_{E1}s_{F1}(92,95)$ $s_{B1}s_{C1}s_{E1}s_{F2}(82,125)$ $s_{B1}s_{C2}s_{E1}s_{F1}(107,75)$**(ns)** $s_{B1}s_{C2}s_{E1}s_{F2}(97,105)$**(d)**	$s_{B2}s_{C1}s_{E1}s_{F1}(102,77)$**(ns)** $s_{B2}s_{C1}s_{E1}s_{F2}(92,107)$**(d)** $s_{B2}s_{C2}s_{E1}s_{F1}(117,57)$**(ns)** $s_{B2}s_{C2}s_{E1}s_{F2}(107,87)$**(ns)**

results in only four optimal instances of path $BCEF$ being stored at node F (all other instances are either dominated, **d**, or not satisfactory, **ns**).

Change 1: Suppose that after processing node E, the world changes and service s_{E2} becomes unavailable. Neglecting this change in our example, will lead to the selection of composite service $s_{B1}s_{C2}s_{E2}s_{F1}$ with the highest utility (lowest price) to perform the requested task, which is an invalid solution (not executable). Instead, composite service $s_{B1}s_{C1}s_{E1}s_{F1}$(ex:92, pr:95) is the optimal solution (the cheapest satisfactory composite service) in this case. Since instance $s_{B1}s_{C1}s_{E1}$ is not recorded at node E (it is dominated by $s_{B2}s_{C1}s_{E2}$), acquiring composite service $s_{B1}s_{C1}s_{E1}s_{F1}$ requires updating the optimal instances of nodes E and F. Hence, to tackle the change that has occurred, nodes C and E need to be reprocessed. Table 5.3 shows the result of revisiting nodes C and E. As can be seen, only two optimal instances are stored in node F, of which, instance $s_{B1}s_{C1}s_{E1}s_{F1}$ is the one with the minimum price.

Change 2: Suppose that after revisiting nodes C and E, the world changes again and

TABLE 5.4: Optimal instances reestimation in response to change 2 (S = start node)

Node	G	C	E	F
validpred	S	SG	SGC	$SGCE$
oinstances (ex,pr)	$s_{G1}(86,5)$ $s_{G2}(26,15)$	$s_{G1}s_{C1}(101,55)$**(ns)** $s_{G1}s_{C2}(116,35)$**(ns)** $s_{G2}s_{C1}(41,65)$ $s_{G2}s_{C2}(56,45)$	$s_{G2}s_{C1}s_{E1}(68,70)$ $s_{G2}s_{C2}s_{E1}(83,50)$	$s_{G2}s_{C1}s_{E1}s_{F1}(98,80)$ $s_{G2}s_{C1}s_{E1}s_{F2}(88,110)$ $s_{G2}s_{C2}s_{E1}s_{F1}(113,60)$**(ns)** $s_{G2}s_{C2}s_{E1}s_{F2}(103,90)$**(ns)**

a new service s_{G2}(ex:26, pr:15) joins the candidate services of task G, thus changing node G's ideal quality value for execution time from 86 to 26. As a result, plan $GCEF$ becomes acceptable with respect to plan-based pruning (i.e. $SPLAN = SPLAN \cup \{GCEF\}$), and its instance $s_{G2}s_{C1}s_{E1}s_{F1}$(ex:98, pr:80) is now a better solution (regarding price) than $s_{B1}s_{C1}s_{E1}s_{F1}$(ex:92, pr:95). Therefore, in order to take advantage of path $GCEF$, which was considered unsatisfactory before the addition of s_{G2}, the valid predecessors of nodes G, C, E, and F should be modified as follows: $validpred(G) = validpred(G) \cup \{S\}$, $validpred(C) = validpred(C) \cup \{SG\}$, $validpred(E) = validpred(E) \cup \{SGC\}$, and $validpred(F) = validpred(F) \cup \{SGCE\}$. The selection algorithm execution should then roll back to the start node *start* so that the optimal instances at nodes G, C, E, and F can be updated. Note that the optimal instances of paths B, BC, BCE and $BCEF$ are not affected by this change, and thus do not have to be recalculated when revisiting nodes *start*, B, C, and E. Table 5.4 shows the optimal instances for the additional valid predecessors at nodes G, C, E, and F after (re)processing edges $(start, G)$, (G, C), (C, E) and (E, F).

5.3 Reactive Selection Algorithm

As illustrated by the above example, in order for the selection algorithm to be reactive (capable of handling changes that occur in the environment), some already visited nodes might require reprocessing so that the changes are reflected in the optimal instances. To keep selection efficient, we need to ensure that only the optimal instances *affected*

by the change are modified when revisiting nodes. This can be achieved by associating the valid predecessors of each node with either a *processed* (prc) or an *unprocessed* (unp) status, which can be formally modelled as a function $status(v, p_v)$, such that:

$$\forall v \in V_{PK}, \forall p_v \in validpred(v), \quad status(v, p_v) \in \{\text{prc}, \text{unp}\}$$

Status $status(v, p_v) = \text{prc}$ indicates that the optimal instances of path $p_v + v$ are already recorded at node v, so there is no need to recompute them when reprocessing edge $(enode(p_v), v)$. Status $status(v, p_v) = \text{unp}$ indicates that the optimal instances of path $p_v + v$ require (re)calculation.

Besides introducing the status function, responding to changes in the environment involves adding the following four steps to the selection algorithm: (i) updating the request-based non-dominated services of the node where the change occurred, (ii) updating the status of the nodes in the plan paths graph so that the affected optimal instances are modified, (iii) updating the valid predecessors of the nodes to reflect the change (if any) in the plans acceptable according to plan-based pruning, and (iv) identifying the node to which the algorithm should roll back. The last step depends on the set of nodes, NTR, to be revisited as a result of the change. This set is specified in the second and third steps, and includes the nodes that require reprocessing in order for the change to be reflected (the node from which to start reprocessing is first in the topological order, among the nodes in NTR). This reactive selection algorithm is provided in Algorithm 4, with the above four steps being summarised in Procedure 6. The first three steps are detailed next, using the following additional notation and functions: α_o and α_n represent α before and after the occurrence of a change, $eservice(ins)$ returns the last service in instance ins, $indexof(p, v)$ returns the index of node v in path p, and $atindex(ins, i)$ returns the service that appears at index i in instance ins.

Algorithm 4 R-Selection-Algorithm

1: generate plan paths graph $g_{PK} = (V_{PK}, E_{PK})$
2: assign to each node $v \in V_{PK} \setminus \{v_{start}\}$ its valid predecessors $validpred(v)$, such that $\forall p_v \in validpred(v), status(v, p_v) = \text{unp}$
3: sort the nodes in V_{PK} topologically
4: store the empty instance at v_{start} and empty service at v_{dest}
5: nextInd $\leftarrow 0$
6: **while** nextInd $< |V_{PK}| - 1$ **do**
7: $v \leftarrow$ the node at position *nextInd* according to the topological order
8: currInd $\leftarrow$ nextInd
9: nextInd $\leftarrow$ nextInd+1
10: **if** $v = v_{start}$ **or** $validpred(v) \neq \emptyset$ **then**
11: $E \leftarrow \{(v, u) \in E_{PK}\}$
12: **while** (**not** $empty(E)$) **and** (nextInd $>$ currInd) **do**
13: $(v, u) \leftarrow$ an element from E
14: $E \leftarrow E \setminus \{(v, u)\}$
15: r-process-edge(v,u,currInd,nextInd)
16: Solution ins_{sol} is the optimal instance at v_{dest} with the highest utility $utility_c$

Procedure 5 r-process-edge(v,u,currInd,nextInd)

1: $P \leftarrow \{p_u \in validpred(u) \mid enode(p_u) = v\}$
2: **while** (**not** $empty(P)$) **and** (nextInd $>$ currInd) **do**
3: $p_u \leftarrow$ an element from P
4: $p_v \leftarrow p_u - v$
5: $P \leftarrow P \setminus \{p_u\}$
6: **if** $status(u, p_u) = \text{unp}$ **then**
7: **for** each $s \in rcandidates(u)$ **do**
8: **for** each optimal instance $ins_v \in oinstances(v, p_v)$ **do**
9: **if** instance $ins_v + s$ is satisfactory **then**
10: check-instance-optimality($ins_v + s$,u,p_u)
11: $status(u, p_u) \leftarrow$ prc
12: **else**
13: do-nothing
14: observe the world
15: **if** change occurred **then**
16: process-changes(nextInd)

5.3.1 The effect on non-dominated services

A change in the available services of task $v \in V_{PK}$ while processing a particular request, such as the addition of a new service, the deletion of an existing service, or changes in the quality values of a service, might affect this task's set of request-based non-dominated services, causing the addition of new services ADD to this set while removing existing

Procedure 6 process-changes(nextInd)

1: $v \leftarrow$ the node where the change occurred
2: $NTR \leftarrow \emptyset$ (the nodes to be reprocessed)
3: $rcandidates(v) \leftarrow (rcandidates(v) \setminus RMV) \cup ADD$
4: **if** $(ADD \neq \emptyset) \ \vee \ (RMV \neq \emptyset)$ **then**
5: update $status(u, p_u)$ of each $p_u \in validpred(u \in V_{PK} \setminus \{v_{start}\})$
6: **if** $\exists a \in AR, \ \ ideal_{t_n}(v, a) \neq ideal_{t_o}(v, a)$ **then**
7: update $validpred(u)$ of each $u \in V_{PK} \setminus \{v_{start}\}$
8: **if** $NTR \neq \emptyset$ **then**
9: nextInd $\leftarrow min($nextInd, the smallest index among the nodes in NTR$)$

ones RMV. That is:

$$rcandidates_n(v) = (rcandidates_o(v) \setminus RMV) \cup ADD$$

where $rcandidates_o(v)$ and $rcandidates_n(v)$ are the request-based non-dominated services of task v before and after the change, respectively.

The definition of sets ADD and RMV depends on the type of change (addition, deletion, or changes in qualities of a service), and thus is specified for each case separately.

5.3.1.1 Addition of a service

Where a new service s_n joins the candidate services of task v, i.e. $candidates_n(v) = candidates_o(v) \cup \{s_n\}$, the following two cases are distinguished. If $\exists s \in rcandidates_o(v)$ such that s r-dm s_n, no change is made to set $rcandidates_o(v)$, i.e. $ADD = RMV = \emptyset$. Otherwise, service s_n is added to set $rcandidates_o(v)$, i.e. $ADD = \{s_n\}$, removing from this set all the services that are request-based dominated by s_n, i.e. $RMV = \{s \in rcandidates_o(v) \mid s_n \text{ r-dm } s\}$.

5.3.1.2 Deletion of a service

Where a candidate service s_o of task v becomes unavailable, i.e. $candidates_n(v) = candidates_o(v) \setminus \{s_o\}$, the following two cases are distinguished. If service s_o is not a member of set $rcandidates_o(v)$, its deletion does not affect this set, i.e. $ADD = RMV = \emptyset$. Otherwise, s_o is removed from $rcandidates_o(v)$, i.e. $RMV = \{s_o\}$, adding to it all task v's candidate services not previously included in this set which, as a result of eliminating s_o, become non-dominated according to the current request, i.e. $ADD = \{s \in dominated_s(s_o) \mid \forall s_i \in (rcandidates_o(v) \setminus \{s_o\}) \cup dominated_s(s_o),\ \neg(s_i \text{ r-dm } s)\}$.

5.3.1.3 Changes in the quality values of a service

Where a candidate service s_o of task v changes its quality values, i.e. $candidates_n(v) = (candidates_o(v) \setminus \{s_o\}) \cup \{s_{ch}\}$, with s_{ch} denoting this service after the change, the following two cases are distinguished. *Case 1*: $s_o \notin rcandidates_o(v)$. Here, if service s_o r-dm s_{ch}, no change to set $rcandidates_o(v)$ is required, i.e. $ADD = RMV = \emptyset$. Otherwise, this case is treated similarly to the addition of a new service $s_n = s_{ch}$. *Case 2*: $s_o \in rcandidates_o(v)$. Here, we have the following three sub-cases.

- *Case 2.1*: s_{ch} r-dm s_o. In this case, the post-change version s_{ch} remains request-based non-dominated. Thus, service s_o is replaced with s_{ch} in set $rcandidates_o(v)$, removing from this set all the services that are request-based dominated by s_{ch}, i.e. $ADD = \{s_{ch}\}$, $RMV = \{s_o\} \cup \{s \in rcandidates_o(v) \setminus \{s_o\},\ \ s_{ch} \text{ r-dm } s\}$.

- *Case 2.2*: s_o r-dm s_{ch}. In this case, the post-change version s_{ch} should be checked for request-based non-dominance, along with all task v's candidate services previously dominated by the pre-change version s_o, leading to sets ADD and RMV similar to those in the deletion case, i.e. $ADD = \{s \in dominated_s(s_o) \mid \forall s_i \in$

$(rcandidates_o(v) \setminus \{s_o\}) \cup dominated_s(s_o),\ \ \neg(s_i \text{ r-dm } s)\},\ \ \ RMV = \{s_o\}$. Note that $s_{ch} \in dominated_s(s_o)$ in this case.

- *Case 2.3*: neither of s_{ch} and s_o request-based dominates the other. In this case, if $\exists s \in rcandidates_o(v) \setminus \{s_o\}$ such that s r-dm s_{ch}, then sets ADD and RMV are defined as in *Case 2.2*. Otherwise, s_{ch} is request-based non-dominated, and thus the services to be added are service s_{ch} along with all task v's candidate services not previously included in $rcandidates_o(v)$ which, as a result of replacing s_o with s_{ch}, become non-dominated according to the current request, i.e. $ADD = \{s_{ch}\} \cup \{s \in dominated_s(s_o) \mid \forall s_i \in (rcandidates_o(v) \setminus \{s_o\}) \cup \{s_{ch}\} \cup dominated_s(s_o),\ \ \neg(s_i \text{ r-dm } s)\}$. The services to be removed are service s_o plus all the services in $rcandidates_o(v) \setminus \{s_o\}$ that are request-based dominated by s_{ch}, i.e. $RMV = \{s_o\} \cup \{s \in rcandidates_o(v) \setminus \{s_o\},\ \ s_{ch} \text{ r-dm } s\}$.

5.3.2 The effect on status function

Altering the request-based non-dominated services of task $v \in V_{PK}$, might lead to corresponding changes in the optimal instances of the paths affected by this alteration, which are those paths containing node v in the plan paths graph. For instance, in the example of Section 5.2, removing service s_{E2} from the request-based non-dominated services of task E has necessitated modifying the optimal instances of paths BCE and $BCEF$. One way to take account of these changes is by *recalculating* the optimal instances of all affected paths. We refer to this as **Recalculation (or First) Status Semantics**. More specifically, each processed valid predecessor of node v, as well as the processed valid predecessors containing node v, are assigned an *unprocessed* status. That is: $\forall u \in V_{PK}, \forall p_u \in validpred(u)$,

$$[(u = v) \vee (v \in nodes(p_u))] \Rightarrow [status_n(u, p_u) = \text{unp}]$$

Since recomputing the optimal instances at node u for valid predecessor p_u requires revisiting node $enode(p_u)$, the last node of each *processed* valid predecessor of node v (the node affected by the change) should be added to set NTR:

$$\forall p_v \in validpred(v), \quad [status_o(v, p_v) = \text{prc}] \Rightarrow [NTR := NTR \cup \{enode(p_v)\}]$$

Finally, the status of all unprocessed valid predecessors, and those not containing node v (whose optimal instances are not affected by the change, and thus do not require modification), remains the same. That is: $\forall u \in V_{PK}, \forall p_u \in validpred(u)$,

$$[(status_o(u, p_u) = \text{unp}) \vee ((u \neq v) \wedge (v \notin nodes(p_u)))] \Rightarrow \\ [status_n(u, p_u) = status_o(u, p_u)]$$

Although the above way of reacting to changes is effective, it causes unnecessary full recalculation of the affected paths' optimal instances. Thus, more efficient change handling can be achieved by ensuring that only the necessary updates are made to these instances, instead of recomputing them from scratch. For example, when a new service joins the request-based non-dominated services of task $v \in V_{PK}$, new instances (that contain the additional service) become available for each path containing node v. Hence, taking account of these additional instances requires checking their optimality against the existing optimal instances of the affected paths, without the need for recalculating the latter from scratch. Similarly, in the case of the deletion of a request-based non-dominated service of task $v \in V_{PK}$, updating the optimal instances of an affected path involves removing all the instances containing the deleted service, in addition to checking the optimality of all those previously dominated by at least one eliminated instance.

To accomplish such behaviour, the status function semantics is updated as follows.

Status $status(u, p_u) = \text{unp}$ indicates that the optimal instances of path $p_u + u$ have not been computed yet. Status $status(u, p_u) = \text{prc}$ indicates that the optimal instances of path $p_u + u$ are already recorded at node u, but might require some modifications specified in terms of three sets $addserv(u, p_u)$, $addins(u, p_u, i \in \mathbb{Z}^+)$, and $domcheck(u, p_u)$, as follows.

- $addserv(u, p_u) \subset candidates(u)$: is an *additional services* function, specifying what services of node u need to be joined with path p_u's optimal instances when updating the optimal instances of path $p_u + u$.
- $addins(u, p_u, i \in \mathbb{Z}^+) \subset S$: is an *additional instances* function, specifying what optimal instances of path p_u need to be joined with node u's services when updating the optimal instances of path $p_u + u$. More specifically, $s \in addins(u, p_u, i)$ indicates that, of the additional optimal instances ins of path p_u to be combined with node u's services, are those containing service s at position i (i.e. $atindex(ins, i) = s$).
- $domcheck(u, p_u) \subset instances(p_u + u)$, is a *domination check* function, specifying which optimal instances of path $p_u + u$ become unavailable. Thus, when updating the optimal instances of path $p_u + u$, all its instances previously dominated by at least one instance in $domcheck(u, p_u)$ should be checked for optimality.

Note that $addserv(u, p_u) = addins(u, p_u, i \in \mathbb{Z}^+) = domcheck(u, p_u) = \emptyset$, if no modifications to the optimal instances of path $p_u + u$ are required, or when $status(u, p_u) = \text{unp}$. Based on this new semantics of the status function, referred to as **Modification (or Second) Status Semantics**, Procedure 7 should replace line 13 of Procedure 5.

Sets *addserv*, *addins*, and *domcheck*, associated with each valid predecessor, are modified each time a change occurs in the environment. This modification depends on the

Procedure 7 Replacement to Line 13 of Procedure 5

if $domcheck(u, p_u) \neq \emptyset$ **then**
  for each $s \in rcandidates(u) \setminus addserv(u, p_u)$ **do**
    for each $ins_v \in oinstances(v, p_v)$ s.t. $\forall i \in \mathbb{Z}^+,\ \ atindex(ins_v, i) \notin addins(u, p_u, i)$ **do**
      if $\exists ins_u \in domcheck(u, p_u),\ \ ins_u$ r-dm $ins_v + s$ **then**
        check-instance-optimality($ins_v + s$,u,p_u)
if $\exists i \in \mathbb{Z}^+,\ \ addins(u, p_u, i) \neq \emptyset$ **then**
  for each $ins_v \in oinstances(v, p_v)$ s.t. $\exists i \in \mathbb{Z}^+,\ \ atindex(ins_v, i) \in addins(u, p_u, i)$ **do**
    for each $s \in rcandidates(u) \setminus addserv(u, p_u)$ **do**
      if instance $ins_v + s$ is satisfactory **then**
        check-instance-optimality($ins_v + s$,u,p_u)
if $addserv(u, p_u) \neq \emptyset$ **then**
  for each $s \in addserv(u, p_u)$ **do**
    for each $ins_v \in oinstances(v, p_v)$ **do**
      if instance $ins_v + s$ is satisfactory **then**
        check-instance-optimality($ins_v + s$,u,p_u)
$addserv(u, p_u) \leftarrow \emptyset$; $addins(u, p_u, i \in \mathbb{Z}^+) \leftarrow \emptyset$; $domcheck(u, p_u) \leftarrow \emptyset$

change type (addition of a service, deletion of a service, or changes in the quality values of a service), and is thus defined for each case separately.

5.3.2.1 Addition of a service

Where a new service s_n joins the candidate services of node $v \in V_{PK}$ such that $s_n \in ADD$, sets *addserv*, *addins*, and *domcheck* of each valid predecessor $p_u \in validpred(u \in V_{PK})$ are adjusted according to the following three cases.

- *Case 1:* p_u is not processed yet, i.e. $status(u, p_u) = \text{unp}$, or is not affected by this addition, i.e. $(u \neq v) \wedge (v \notin nodes(p_u))$. In this case, no change is made to the sets *addserv*, *addins*, and *domcheck* associated with p_u.
- *Case 2:* p_u is a processed valid predecessor of node v, i.e.

$$(status(u, p_u) = \text{prc}) \ \wedge \ (u = v)$$

In this case, only set $addserv(u, p_u)$ is modified, by adding to it the new service s_n while removing all the services belonging to RMV:

$$addserv_n(u, p_u) = (addserv_o(u, p_u) \setminus RMV) \cup ADD$$

Additionally, all existing optimal instances ending with a service in RMV are eliminated:

$$oinstances_n(u, p_u) = oinstances_o(u, p_u) \setminus \{ins \in oinstances_o(u, p_u) \mid eservice(ins) \in RMV \setminus addserv_o(u, p_u)\}$$

Finally, to allow the update of the optimal instances, node $enode(p_u)$ is added to the set of nodes to be revisited, i.e. $NTR := NTR \cup \{enode(p_u)\}$.

- *Case 3:* p_u is a processed valid predecessor containing node v at position i, i.e.

$$(status(u, p_u) = \text{prc}) \wedge (v \in nodes(p_u)) \wedge (indexof(p_u, v) = i)$$

In this case, only set $addins(u, p_u, i)$ is modified, by adding to it the new service s_n while removing all the services that are members of RMV:

$$addins_n(u, p_u, i) = (addins_o(u, p_u, i) \setminus RMV) \cup ADD$$

Like the previous case, all the optimal instances containing a service from RMV at position i are eliminated:

$$oinstances_n(u, p_u) = oinstances_o(u, p_u) \setminus \{ins \in oinstances_o(u, p_u) \mid atindex(ins, i) \in RMV \setminus addins_o(u, p_u, i)\}$$

5.3.2.2 Deletion of a service

Where a request-based non-dominated service s_o of node $v \in V_{PK}$ becomes unavailable, sets *addserv*, *addins*, and *domcheck* of each valid predecessor $p_u \in validpred(u \in V_{PK})$ are adjusted according to the following five cases.

- *Case 1:* p_u is not processed yet, i.e. $status(u, p_u) = \text{unp}$, or is not affected by this deletion, i.e. $(u \neq v) \wedge (v \notin nodes(p_u))$. In this case, no change is made to the sets *addserv*, *addins*, and *domcheck* associated with p_u.

- *Case 2:* p_u is a processed valid predecessor of node v, and the eliminated service is a member of $addserv(u, p_u)$, i.e.

$$(status(u, p_u) = \text{prc}) \ \wedge \ (u = v) \ \wedge \ (s_o \in addserv_o(u, p_u))$$

  In this case, only set $addserv(u, p_u)$ is modified, by adding to it all the services in ADD while eliminating the deleted service s_o:

$$addserv_n(u, p_u) = (addserv_o(u, p_u) \setminus RMV) \cup ADD$$

  Here, there is no need to add node $enode(p_u)$ to set NTR, since valid predecessor p_u had not been reprocessed before the occurrence of the current change $(addserv_o(u, p_u) \neq \emptyset)$.

- *Case 3:* p_u is a processed valid predecessor of node v, and the eliminated service is not a member of $addserv(u, p_u)$, i.e.

$$(status(u, p_u) = \text{prc}) \ \wedge \ (u = v) \ \wedge \ (s_o \notin addserv_o(u, p_u))$$

In this case, all the existing optimal instances ending with service s_o, denoted IE, are eliminated and added to set $domcheck(u, p_u)$:

$$oinstances_n(u, p_u) = oinstances_o(u, p_u) \setminus IE \quad \wedge$$
$$domcheck_n(u, p_u) = domcheck_o(u, p_u) \cup IE$$

where $IE = \{ins \in oinstances_o(u, p_u) \mid eservice(ins) \in RMV\}$. Additionally, the end node of p_u is regarded as a node to be revisited, i.e. $NTR := NTR \cup \{enode(p_u)\}$.

- *Case 4:* p_u is a processed valid predecessor containing node v at position i, and the eliminated service is a member of $addins(u, p_u, i)$, i.e.

$$(status(u, p_u) = \text{prc}) \ \wedge \ (v \in nodes(p_u)) \ \wedge \ (indexof(p_u, v) = i) \wedge (s_o \in addins_o(u, p_u, i))$$

In this case, only set $addins(u, p_u, i)$ is modified, by adding to it all the services in ADD, while eliminating the deleted service s_o:

$$addins_n(u, p_u, i) = (addins_o(u, p_u, i) \setminus RMV) \cup ADD$$

- *Case 5:* p_u is a processed valid predecessor containing node v at position i, and the eliminated service is not a member of $addins(u, p_u, i)$, i.e.

$$(status(u, p_u) = \text{prc}) \ \wedge \ (v \in nodes(p_u)) \ \wedge \ (indexof(p_u, v) = i) \wedge (s_o \notin addins_o(u, p_u, i))$$

In this case, all the existing optimal instances containing service s_o at position i, denoted IAI, are eliminated and added to set $domcheck(u, p_u)$:

$$oinstances_n(u, p_u) = oinstances_o(u, p_u) \setminus IAI \quad \wedge$$
$$domcheck_n(u, p_u) = domcheck_o(u, p_u) \cup IAI$$

where $IAI = \{ins \in oinstances_o(u, p_u) \mid atindex(ins, i) \in RMV\}$.

5.3.2.3 Changes in the quality values of a service

Changes in the quality values of a request-based non-dominated service s_o of node $v \in V_{PK}$, with s_{ch} denoting this service after the change, can be defined in terms of the deletion and addition of a service, as follows. If service s_o r-dm s_{ch}, or service $s_{ch} \notin ADD$, this case is modelled as the deletion of service s_o with $ADD_{del} = ADD$, $RMV_{del} = RMV$. Therefore, the same updates to the sets *addserv*, *addins*, and *domcheck* in the deletion case are applied here. Otherwise, this case is treated similarly to the deletion of a request-based non dominated service s_o with $ADD_{del} = ADD \setminus \{s_{ch}\}$, $RMV_{del} = \{s_o\}$, followed by the subsequent addition of service s_{ch} with $ADD_{add} = \{s_{ch}\}$, $RMV_{add} = RMV \setminus \{s_o\}$.

5.3.3 The effect on valid predecessors

A modification in the request-based non-dominated services of a task $v \in V_{PK}$, might affect this task's ideal quality values for the constrained attributes and, as a result, might alter the set of plans $SPLAN$ to be considered for composition according to plan-based pruning. For instance, the addition of service s_{G2} to task G's services in the example of Section 5.2, causes the addition of a new plan (plan $GCEF$) to set $SPLAN$.

Assuming that as a consequence of a change, the ideal quality values of node v for the constrained attributes are modified, i.e. $\exists a \in AR, \quad ideal_{tn}(v,a) \neq ideal_{to}(v,a)$, the valid predecessors of each node $u \in V_{PK}$ should be updated as follows. Given a path $p_u + u$ from the start node to u, path p_u is considered a valid predecessor of node u after the change, i.e. $validpred(u) := validpred(u) \cup \{p_u\}$, if there exists at least one path p_i from u to the destination node such that $p_u + p_i \in SPLAN_n$ (the set of plans acceptable according to plan-based pruning after the change).

Note that, when p_u is a new valid predecessor of node u (it is not regarded as node u's valid predecessor before the change), node $enode(p_u)$ should be (re)visited in order for the optimal instances of path $p_u + u$ to be calculated:

$$NTR := NTR \cup \begin{cases} \emptyset & \text{if } p_u \in validpred_o(u) \\ enode(p_u) & \text{otherwise} \end{cases}$$

Alternatively, if each path p_i from u to the destination node satisfies: $p_u + p_i \notin SPLAN_n$, path p_u is eliminated from the valid predecessors of node u (when it exists before the change), removing all the optimal instances of path $p_u + u$ stored at this node, i.e. $validpred(u) := validpred(u) \setminus \{p_u\}$.

Finally, the status of all newly added valid predecessors is set to *unprocessed*:

$$\forall u \in V_{PK}, \forall p_u \in validpred_n(u) \setminus validpred_o(u), \quad status(u, p_u) = \text{unp}$$

5.4 Analytical Study

The goal of this section is to analyse the time complexity of the proposed reactive selection algorithm, and compare it with that of the static selection (which ignores

selection-time changes). For simplicity, the analysis is provided for the specific case of one abstract plan.

Responding to a change during service selection may require recomputing (or updating) the optimal instances of some already processed nodes, thus causing additional overhead to the selection process. For a sequential abstract plan, the overhead added by reacting to a service change at node v_{ch} while processing node v_{pr} such that $pr \geq ch$, is as follows:

$$overhead = \sum_{i=ch}^{pr} \tau(oinstances_n(v_i))$$

where $\tau(oinstances_n(v_i))$ is the time complexity of recomputing (or updating) the optimal instances at node v_i, as a result of the change.

In what follows, we analyse and compare such overhead for the two proposed reactive selection versions corresponding to the *recalculation* and *modification* status semantics (see Section 5.3.2). Note that the focus is on the step of optimal instances recalculation (modification) in response to the change. This is because, when compared to this step, the time required for the other change processing steps (i.e. updating the request-based non-dominated services of node v_{ch}, and updating the status of affected nodes) is negligible.

5.4.1 Recalculation status semantics

According to the recalculation (first) semantics, *fs*, of the status function, all affected optimal instances should be recalculated from scratch (see Lines 7-10 of Procedure 5). For a node affected by the change, $v_{i \geq ch}$, the time complexity of recomputing the optimal instances, $\tau^{fs}(oinstances_n(v_i))$, is as follows:

$$\tau^{fs}(oinstances_n(v_{i \geq ch})) \text{ is } O((|oinstances_n(v_{i-1})| \times |rcandidates_n(v_i)|)^2) \quad (5.1)$$

with:

$$\forall i < ch, \quad |oinstances_n(v_i)| = |oinstances_o(v_i)| \quad \wedge$$
$$\forall i \neq ch, \quad |rcandidates_n(v_i)| = |rcandidates_o(v_i)|$$

From Equation 4.4, and since $|rcandidates_n(v_{ch})| = n' \times sr_{ch}$, we conclude that:

$$|oinstances_n(v_{i \geq ch})| = n^{i-1} \times n' \times \prod_{m=1}^{i} sr_m \times \prod_{m=2}^{i} ir_m \tag{5.2}$$

where $n' = |candidates_n(v_{ch})|$, i.e. $n' = n + 1$ in case of service addition; $n' = n - 1$ in case of service deletion; and $n' = n$ in case of service quality changes.

Consequently, we can distinguish the following two cases for Equation 5.1:

– $\tau^{fs}(oinstances_n(v_{ch}))$ is $O(([n^{ch-1} \times \prod_{m=1}^{ch-1} sr_m \times \prod_{m=2}^{ch-1} ir_m] \times [n' \times sr_{ch}])^2)$
is $O(n^{2ch-2} \times n'^2 \times \prod_{m=1}^{ch} sr_m^2 \times \prod_{m=2}^{ch-1} ir_m^2)$

– $\tau^{fs}(oinstances_n(v_{i>ch}))$ is $O(([n^{i-2} \times n' \times \prod_{m=1}^{i-1} sr_m \times \prod_{m=2}^{i-1} ir_m] \times [n \times sr_i])^2)$
is $O(n^{2i-2} \times n'^2 \times \prod_{m=1}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2)$

The above two cases can thus be generalised to:

$$\tau^{fs}(oinstances_n(v_{i \geq ch})) \text{ is } O(n^{2(i-1)} \times n'^2 \times \prod_{m=1}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2) \tag{5.3}$$

5.4.2 Modification status semantics

According to the modification (second) semantics, *ss*, of the status function, only necessary updates are made to the affected optimal instances, without recalculating those

instances from scratch (see Procedure 7). The updates depend on the type of change that has occurred (addition, deletion, or modification in the quality values of a service). The addition and deletion cases are analysed next, while the modification case can be derived from these two cases.

5.4.2.1 Addition of a Service

Suppose service s_n joins the request-based non-dominated services of node v_{ch}. For node v_{ch}, responding to this change involves combining the optimal instances of node v_{ch-1} with the new service s_n, and then checking the optimality of the resulting combinations against the optimal instances already recorded at node v_{ch}, i.e.

$$\begin{aligned}\tau^{ss}(oinstances_n(v_{ch})) \text{ is } & O(|oinstances_o(v_{ch-1})| \times |oinstances_o(v_{ch})|) \\ \text{is } & O([n^{ch-1} \times \prod_{m=1}^{ch-1} sr_m \times \prod_{m=2}^{ch-1} ir_m] \times [n^{ch} \times \prod_{m=1}^{ch} sr_m \times \prod_{m=2}^{ch} ir_m]) \\ \text{is } & O(n^{2ch-1} \times \prod_{m=1}^{ch-1} sr_m^2 \times \prod_{m=2}^{ch-1} ir_m^2 \times sr_{ch} \times ir_{ch}) \end{aligned} \quad (5.4)$$

(where $|oinstances_o(v_i)| = n^i \times \prod_{m=1}^{i} sr_m \times \prod_{m=2}^{i} ir_m$, according to Equation 4.4)

For node v_i such that $ch < i \leq pr$, updating $oinstances(v_i)$ involves checking the optimality of the newly available instances (obtained by joining the optimal instances containing service s_n at node v_{i-1} , $oinstances_n(v_{i-1})^{s_n}$, with node v_i's services), against

those already computed at node v_i, i.e. $\tau^{ss}(oinstances_n(v_i))$

$$\begin{aligned}
&\text{is } O(\ |oinstances_n(v_{i-1})^{s_n}| \times |rcandidates_n(v_i)| \times |oinstances_n(v_i)|) \\
&\text{is } O(\ [\frac{n^{i-2} \times (n+1) \times \prod_{m=1}^{i-1} sr_m \times \prod_{m=2}^{i-1} ir_m}{(n+1) \times sr_{ch}}] \times [n \times sr_i] \times \\
&\qquad [n^{i-1} \times (n+1) \times \prod_{m=1}^{i} sr_m \times \prod_{m=2}^{i} ir_m]) \\
&\text{is } O(\ n^{2(i-1)} \times (n+1) \times \prod_{\substack{m=1 \\ m \neq ch}}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2 \times sr_{ch} \times ir_i)
\end{aligned} \tag{5.5}$$

(here we assume $|oinstances_n(v_{i-1})^{s_n}| = \frac{|oinstances_n(v_{i-1})|}{|rcandidates_n(v_{ch})|}$, where $|oinstances_n(v_{i \geq ch})|$ is defined by Equation 5.2).

5.4.2.2 Deletion of a Service

Suppose a request-based non-dominated service s_o of node v_{ch} becomes unavailable. For any affected node v_i such that $ch \leq i \leq pr$, responding to this change involves identifying (*idntf*) the candidate instances dominated by at least one optimal instance to be removed (i.e. containing the deleted service s_o), and then checking the optimality (*optchk*) of these identifed instances against the ones already recorded at node v_i, i.e.

$$\tau^{ss}(oinstances_n(v_i)) = \tau(\text{idntf}) + \tau(\text{optchk}) \tag{5.6}$$

To identify the dominated instances of interest, all the candidate instances at node v_i need be compared with those to be removed from this node, $oinstances_o(v_i)^{s_o}$. Hence, the time complexity τ(idntf) is given as follows:

$$\tau(\text{idntf}) \text{ is } O(|oinstances_n(v_{i-1})| \times |rcandidates_n(v_i)| \times |oinstances_o(v_i)^{s_o}|) \tag{5.7}$$

The following two cases can be distinguished for Equation 5.7:

$$
\begin{aligned}
-\ v_{ch}: \quad \tau(\text{idntf}) \quad \text{is} \ & O([n^{ch-1} \times \prod_{m=1}^{ch-1} sr_m \times \prod_{m=2}^{ch-1} ir_m] \times [(n-1) \times sr_{ch}] \times \\
& \frac{n^{ch} \times \prod_{m=1}^{ch} sr_m \times \prod_{m=2}^{ch} ir_m}{n \times sr_{ch}}) \\
\text{is} \ & O(n^{2ch-2} \times (n-1) \times \prod_{m=1}^{ch-1} sr_m^2 \times \prod_{m=2}^{ch-1} ir_m^2 \times sr_{ch} \times ir_{ch}) \\
-\ v_{i>ch}: \quad \tau(\text{idntf}) \quad \text{is} \ O(\ & [n^{i-2} \times (n-1) \times \prod_{m=1}^{i-1} sr_m \times \prod_{m=2}^{i-1} ir_m] \times [n \times sr_i] \times \\
& \frac{n^{i} \times \prod_{m=1}^{i} sr_m \times \prod_{m=2}^{i} ir_m}{n \times sr_{ch}}) \\
\text{is} \ & O(n^{2i-2} \times (n-1) \times \prod_{\substack{m=1 \\ m \neq ch}}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2 \times sr_{ch} \times ir_i)
\end{aligned}
$$

(here we assume $|oinstances_o(v_i)^{s_o}| = \frac{|oinstances_o(v_i)|}{|rcandidates_o(v_{ch})|}$).

The above two cases can thus be generalised to:

$$
\tau(\text{idntf}) \text{ is } O(n^{2i-2} \times (n-1) \times \prod_{\substack{m=1 \\ m \neq ch}}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2 \times sr_{ch} \times ir_i) \tag{5.8}
$$

Now, assume for simplicity that, of $|oinstances(v_{i-1})| \times |rcandidates(v_i)|$ candidate instances at node v_i, the number of those request-based dominated by any optimal (non dominated) instance is: $\frac{|oinstances(v_{i-1})| \times |rcandidates(v_i)|}{|oinstances(v_i)|} = \frac{1}{ir_i}$ (see Equation 4.3). Based on this, $\frac{1}{ir_i} \times |oinstances_o(v_i)^{s_o}|$ candidate instances at node v_i are identified as dominated by the optimal instances to be eliminated. Hence, the time

required for checking these instances' optimality, $\tau(\text{optchk})$, is as follows:

$$\tau(\text{optchk}) \text{ is } O(\frac{1}{ir_i} \times |oinstances_o(v_i)^{s_o}| \times |oinstances_n(v_i)|)$$

$$\text{is } O(\frac{1}{ir_i} \times \frac{n^i \times \prod_{m=1}^{i} sr_m \times \prod_{m=2}^{i} ir_m}{n \times sr_{ch}} \times [n^{i-1} \times (n-1) \times \prod_{m=1}^{i} sr_m \times \prod_{m=2}^{i} ir_m])$$

$$\text{is } O(n^{2i-2} \times (n-1) \times \prod_{\substack{m=1 \\ m \neq ch}}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2 \times sr_{ch} \times ir_i) \quad (5.9)$$

From Equations 5.8 and 5.9, $\tau^{ss}(oinstances_n(v_i))$ (see Equation 5.6) becomes as follows:

$$\tau^{ss}(oinstances_n(v_i)) \text{ is } O(2 \times n^{2i-2} \times (n-1) \times \prod_{\substack{m=1 \\ m \neq ch}}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2 \times sr_{ch} \times ir_i) \quad (5.10)$$

5.4.3 Comparison

To analyse the gain in efficiency achieved by the modification status semantics, let's make again the simplifying assumption that $\forall i,\ sr_i = ir_i = r$. As a result, Equations 5.3, 5.4, 5.5, and 5.10 become as follows.

Recalculation status semantics:

$\tau^{fs}(oinstances_n(v_{i \geq ch}))$ is $O(n^{2i-2} \times (n+1)^2 \times r^{4i-4})$

Modification status semantics:

Addition Case:

$\tau^{ss}(oinstances_n(v_{ch}))$ is $O(n^{2ch-1} \times r^{4ch-4})$

$\tau^{ss}(oinstances_n(v_{i > ch}))$ is $O(n^{2i-2} \times (n+1) \times r^{4i-4})$

Deletion Case:

$\tau^{ss}(oinstances_n(v_{i \geq ch}))$ is $O(2 \times n^{2i-2} \times (n-1) \times r^{4i-4})$

Based on this, comparing time complexities τ^{fs} and τ^{ss}, leads to the following.

Addition Case:

- $\dfrac{\tau^{fs}(oinstances_n(v_{ch}))}{\tau^{ss}(oinstances_n(v_{ch}))} = \dfrac{(n+1)^2}{n}$
- $\dfrac{\tau^{fs}(oinstances_n(v_{i>ch}))}{\tau^{ss}(oinstances_n(v_{i>ch}))} = n+1$

Deletion Case:

- $\dfrac{\tau^{fs}(oinstances_n(v_{i\geq ch}))}{\tau^{ss}(oinstances_n(v_{i\geq ch}))} = \dfrac{(n+1)^2}{2\times(n-1)}$

In other words, the proposed modification status semantics reduces the overhead of regenerating the optimal instances of an already processed node v_i, in response to a change, by a factor of $O(n)$. Although this efficiency gain is expected to decrease when node v_i experiences *accumulated* changes prior to its reprocessing, the gain remains significant as long as the candidate instances at this node, $oinstances(v_{i-1}) \times rcandidates(v_i)$, before and after such changes, are still sufficiently close. It is plausible to assume that the latter holds in the majority of cases, since drastic deviations in the candidate instances require high frequencies of service changes affecting the *same node* and altering *severely* its non-dominated service set, all within the period before reprocessing: an unlikely combination even in very dynamic service-based systems.

5.5 Empirical Study

In this section, we present an experimental evaluation of our reactive selection algorithm, focusing on performance, in terms of execution time, and gain in utility achieved by reacting to changes during selection, with the experimental settings being the same as in Section 4.5.

To evaluate performance, four selection algorithms were compared in terms of time: static selection (*s-alg*), which ignores changes during selection; reactive selection with the first status semantics (*r-fs-alg*); reactive selection with the second status semantics (*r-ss-alg*), which utilises sets *addserv*, *addins* and *domcheck*; and replanning from scratch (*rpl-alg*), which restarts static selection whenever a change occurs. The time of each was averaged over 20 different random requests, and 20 different graph instances, each containing 16 alternative abstract plans, with up to 16 tasks per plan.

Figures 5.1(a), 5.1(b), and 5.1(c) show the running time of the algorithms (with respect to the number of candidate services per task) in the case where a single change occurs during selection, with each figure corresponding to a different change type (addition, deletion, or changes in the qualities of a request-based non-dominated service for a randomly selected node). The results indicate that both *r-fs-alg* and *r-ss-alg* significantly outperform *rpl-alg* which requires almost twice the time required by *r-ss-alg* to produce the same optimal solution. Moreover, as expected, *r-ss-alg* performs better than *r-fs-alg*, especially as the number of candidates grows, and is only slightly less efficient than the static algorithm *s-alg*. The same observations are made in Figure 5.1(d), where execution time corresponds to three changes per selection (addition, deletion, and changes in qualities).

To further compare the performance of *r-fs-alg* and *r-ss-alg* in relation to the number of changes during selection, the number of candidates per task was fixed at 1000, while the number of changes varied between 1 and 5 (change types and locations were selected randomly). As shown in Figure 5.1(e), the efficiency of both algorithms decreases with the increasing environment dynamism, since more rollbacks are required as change increases. We can also observe that *r-ss-alg* performs better than *r-fs-alg* in all cases.

To evaluate utility gain, *s-alg* and *r-ss-alg* were compared in terms of solution *optimality*, estimated as $\frac{utility_{c_{act}}}{utility_{c_{opt}}}$, where $utility_{c_{act}}$ and $utility_{c_{opt}}$ denote the algorithm's

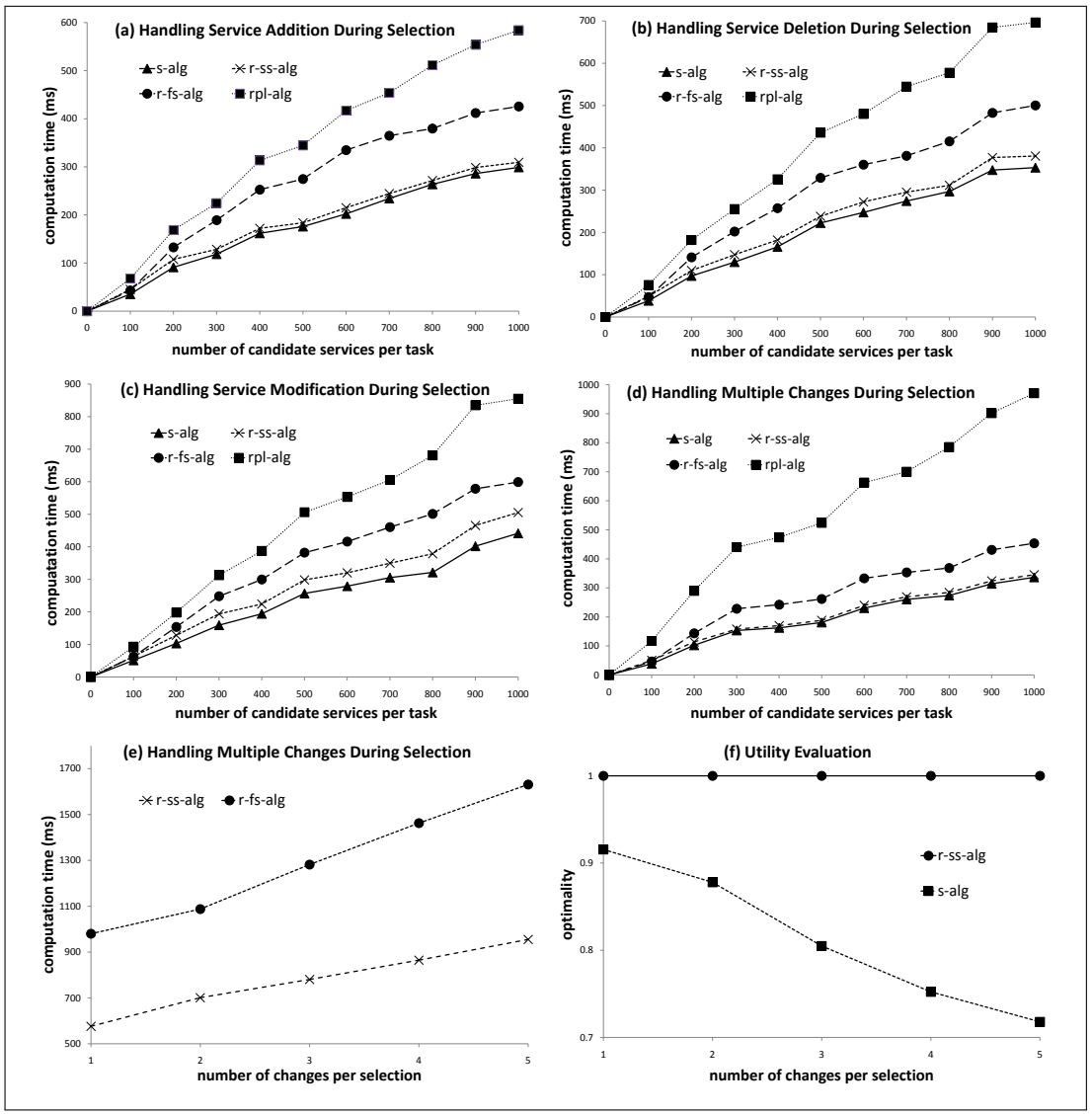

FIGURE 5.1: Evaluating the reactive selection algorithm

actual utility and the optimal utility, respectively. Notice that the actual utility of *s-alg*'s solution p_{salg} may differ from that estimated, since some of p_{salg}'s component services might become unavailable, or have changed their quality values (which might result in the quality constraints being no longer satisfied). Therefore, the actual utility of *s-alg* is calculated as follows. If $\exists a \in AR$ s.t. $value_{cch}(p_{salg}, a) > const_r(a)$, or $\exists s \in nodes(p_{salg})$ s.t. s is no longer available, $utility_{cact}(p_{salg})$ is set to 0. Otherwise,

$utility_{c_{act}}(p_{salg}) = utility_{c_{ch}}(p_{salg})$. Here, $value_{c_{ch}}$ and $utility_{c_{ch}}$ are the quality value and utility functions considering the changes in the environment.

Figure 5.1(f) shows the optimality achieved by static and reactive selection, averaged over 20 graph instances and 20 requests. The number of candidate services per task was fixed at 500, while the changes during selection were varied from 1 to 5. As expected, static selection retrieves a less optimal solution, especially as the number of changes increases.

5.6 Conclusion

In this chapter, we have presented a novel reactive selection algorithm capable of handling changes to services during the selection process, thus ensuring that the selected composition is always valid and meets user quality of service requirements and preferences. Based on the notion of request-based dominance, the algorithm identifies whether a change could have an impact on the optimal solution for the current request, and makes a corresponding adjustment to the affected part of the search graph according to the type of change. To maintain efficiency, the adjustment is achieved through applying a minimal number of modifications to repair the previously generated optimal instances, without recalculating these instances from scratch. The results show that, with only a small overhead, the algorithm is able to find the best solution in dynamic environments, in contrast to static algorithms whose ability to find an optimal solution decreases with an increasing number of changes.

A main assumption of the algorithm is the ability to discover changes when these occur in the environment. This could be achieved, for instance, through periodic testing of the service landscape to detect changes as soon as possible, or via other proactive change detection techniques (see Section 2.5.2.1), but is out of the scope of this thesis.

Moreover, it is worth noting that, in order to keep our selection model generic and domain independent, we abstract out some of the details specific to particular application domains. For example, it might be the case that reservation fees apply when reserving services for tasks in advance. Such fees would thus be lost if tasks are reallocated to other services in response to changes. Although not currently addressed, this could be incorporated into our reactive selection by adding the cost of unkept reservations to the total cost of alternative ones.

The proposed reactive selection algorithm can also act as a basis for efficient execution-time adaptivity, which is detailed in the next chapter.

Chapter 6

Addressing Changes during Service Execution

6.1 Introduction

Although responding to changes in services while performing service selection ensures that the selected composite service is the best possible prior to execution, changes in services can still occur during the execution process, and may possibly lead to the selected solution no longer being available, satisfactory, or optimal. Again, such changes can be either intentional (e.g., a service provider announces improved quality values to increase the competitiveness of its service), or unintentional (e.g., a component service is unavailable due to server or network problems).

As stated earlier, most current service composition approaches handle service changes by monitoring the behaviour of the selected services during execution, so that whenever service unavailability or contract violations are observed, a time-consuming re-selection is performed for the non-executed part of the workflow. In other words, even if service

changes occur at an early execution stage, they are only handled after faulty or quality-violating services are executed [94], resulting in undesired effects at execution time, such as reduced performance due to costly replanning and, in some cases, an inability to find a satisfactory recovery since services have been executed.

In response, this chapter presents a reactive execution algorithm capable of adapting efficiently to service changes during the execution process, with the following features.

First, reaction to changes is performed as soon as they occur during execution, without delaying the reaction until the faulty behavior is executed. This is done in parallel with the execution process, thus minimising interruption time (since re-selection is most likely to finish before the invocation of the next component service), while ensuring that the services re-selected for the remaining non-executed tasks form the best possible solution, given the already executed tasks and the current state of services in the environment.

Second, the execution-time re-selection algorithm is efficient in terms of ensuring that only a small number of modifications are made to react to changes, without the need for performing the re-selection process from scratch.

Third, as opposed to existing approaches where the re-selection is only triggered for repair purposes, our adaptation is also initiated to perform optimisation actions (for instance, in response to the availability of new, better services), thus producing the best solution possible regarding optimality.

Finally, an analysis and classification of changes is provided in order to identify their importance and urgency, and guide the behavior of the executing system correspondingly. For example, the changes that do not currently affect the solution composite service should not cause any interruption in execution, even if the adaptation performed in response to these changes is not completed before the current component service finishes

its execution (i.e. this adaptation will be continued during the execution of the next component service).

The rest of the chapter is organised as follows. Section 6.2 presents a motivating example for reactive execution, followed by our reactive execution algorithm in Section 6.3. A classification of changes is introduced in Section 6.4, based on which the reactive behaviour of the system is analysed in Section 6.5. The proposed algorithm is evaluated analytically and empirically in Sections 6.6 and 6.7, respectively. Finally, Section 6.8 concludes the chapter.

6.2 Motivating Example

To illustrate the need for reactive service execution capable of handling changes in services during the execution process, consider the same example of Section 5.2, in which the user has issued a request to *plan holiday*, and is interested in minimising the price while satisfying the constraint $ex \leq 100$. The plan paths graph for the requested task, and the request-based non-dominated candidate services as well as the valid predecessors of the involved sub-tasks, are as provided in Figure 4.2, and Table 5.1, respectively. The optimal solution for the user is instance $s_{B1}s_{C2}s_{E2}s_{F1}$ which has the lowest price (see Table 5.2). In what follows, we give examples of changes that might occur while executing the selected composite service $s_{B1}s_{C2}s_{E2}s_{F1}$.

6.2.1 Addition of a service

Scenario 1: suppose that while executing s_{B1} of composite service $s_{B1}s_{C2}s_{E2}s_{F1}$, a new service $s_{C3}(\text{ex:}20, \text{pr:}35)$ joins the candidate services of task C. Consequently, four additional combinations are possible from this point (see Table 6.1), of which composite

TABLE 6.1: Additional service combinations as a result of adding service s_{C3}

Composite Service	(ex, pr)
$s_{B1}s_{C3}s_{E1}s_{F1}$	$(97, 80)$
$s_{B1}s_{C3}s_{E1}s_{F2}$	$(87, 110)$
$s_{B1}s_{C3}s_{E2}s_{F1}$	$(85, 95)$
$s_{B1}s_{C3}s_{E2}s_{F2}$	$(75, 125)$

TABLE 6.2: Additional service combinations as a result of adding service s_{D1}

Composite Service	(ex, pr)
$s_{B1}s_{C1}s_{D1}$	$(75, 90)$
$s_{B1}s_{C2}s_{D1}$	$(90, 70)$

service $s_{B1}s_{C3}s_{E1}s_{F1}$(ex:97, pr:80) produces a more optimal (cheaper) solution than the currently selected composition $s_{B1}s_{C2}s_{E2}s_{F1}$(ex:95, pr:90).

Scenario 2: suppose that while executing s_{B1} of composite service $s_{B1}s_{C2}s_{E2}s_{F1}$, a new service s_{D1}(ex:40, pr:10) joins the candidate services of task D. As a result, plan BCD becomes acceptable according to plan-based pruning, and two additional instances of this plan are satisfactory from this point (see Table 6.2), of which composite service $s_{B1}s_{C2}s_{D1}$(ex:90, pr:70) is better than the currently selected composition $s_{B1}s_{C2}s_{E2}s_{F1}$(ex:95, pr:90) regarding both price and execution time.

6.2.2 Deletion of a service

Scenario 3: suppose that while executing s_{B1} of composite service $s_{B1}s_{C2}s_{E2}s_{F1}$, service s_{C2} becomes unavailable. Here, simply replacing s_{C2} with s_{C1} will result in composition $s_{B1}s_{C1}s_{E2}s_{F1}$(ex:80, pr:110), which is not optimal regarding the price. Hence, both s_{C2} and s_{E2} should be substituted in this case in order to obtain the new optimal satisfactory solution, which is service $s_{B1}s_{C1}s_{E1}s_{F1}$(ex:92, pr:95).

TABLE 6.3: New aggregated quality values as a result of changing the quality values of service s_{C2} to (ex:50,pr:20)

Composite Service	(ex, pr)
$s_{B1}s_{C1}s_{E1}s_{F1}$	$(92, 95)$
$s_{B1}s_{C1}s_{E1}s_{F2}$	$(82, 125)$
$s_{B1}s_{C1}s_{E2}s_{F1}$	$(80, 110)$
$s_{B1}s_{C1}s_{E2}s_{F2}$	$(70, 140)$
$s_{B1}s_{C2}s_{E1}s_{F1}$	$(127, 65)$
$s_{B1}s_{C2}s_{E1}s_{F2}$	$(117, 95)$
$s_{B1}s_{C2}s_{E2}s_{F1}$	$(115, 80)$
$s_{B1}s_{C2}s_{E2}s_{F2}$	$(105, 110)$

TABLE 6.4: New aggregated quality values as a result of changing the quality values of service s_{C2} to (ex:20,pr:30)

Composite Service	(ex, pr)
$s_{B1}s_{C2}s_{E1}s_{F1}$	$(97, 75)$
$s_{B1}s_{C2}s_{E1}s_{F2}$	$(87, 105)$
$s_{B1}s_{C2}s_{E2}s_{F1}$	$(85, 90)$
$s_{B1}s_{C2}s_{E2}s_{F2}$	$(75, 120)$

6.2.3 Changes in the quality values of a service

Scenario 4: suppose that while executing s_{B1} of composite service $s_{B1}s_{C2}s_{E2}s_{F1}$, service s_{C2} changes its quality values to (ex:50, pr:20). From this point, the selected composite service is no longer satisfactory, and the new optimal satisfactory one is $s_{B1}s_{C1}s_{E1}s_{F1}$ (see Table 6.3).

Scenario 5: suppose that after executing services s_{B1} and s_{C2} of composite service $s_{B1}s_{C2}s_{E2}s_{F1}$, s_{C2} performs better than expected with the quality values (ex:20, pr:30). In this case, the selected composite service is no longer optimal, and the new optimal one is $s_{B1}s_{C2}s_{E1}s_{F1}$ (see Table 6.4).

6.3 Reactive Service Execution

As stated earlier, in most current composition approaches, quality violations and faulty behaviour of services are detected only after their occurrence (i.e. after executing the corresponding service), resulting in undesired effects at execution time, such as reduced performance due to the high re-selection overhead and, in some cases, inability to find a satisfactory solution given the already executed services. For instance, although service s_{C2} in Scenario 3 is deleted during the execution of service s_{B1}, its unavailability is only observed when trying to invoke this service, causing the composite service execution to stop until re-selection is performed. Similarly, in Scenario 4, detecting the changes in the quality values of service s_{C2} after its execution results in an unrecoverable situation, since no satisfactory solution can be found from this point (all the service combinations including services s_{B1} and s_{C2} violate the user's execution time constraint).

To tackle the above problems, adaptation to changes should be performed as soon as these changes occur in the environment, concurrently with the execution of the current service, thus reducing delay between service executions, and increasing the chance of a successful recovery. For instance, in Scenario 3, re-selecting services for tasks C, E and F in response to the deletion of service s_{C2} can be achieved while executing service s_{B1}, without causing delay to the composite service execution. Moreover, the adaptation algorithm should be *efficient*, since a costly re-selection from scratch can still cause an interruption to execution, even if it is triggered while the current service is still running.

6.3.1 Reverse plan paths graph

To achieve efficiency, we aim to react to execution-time changes in a similar manner to our reactive selection, by applying only the necessary updates to the optimal instances

TABLE 6.5: Optimal instances of node F after s_{B1} execution

Node	F
validpred	$SBCE$
oinstances (ex,pr)	$s_{B1}s_{C1}s_{E1}s_{F1}(92, 95)$
	$s_{B1}s_{C1}s_{E2}s_{F1}(80, 110)$
	$s_{B1}s_{C1}s_{E2}s_{F2}(70, 140)$
	$s_{B1}s_{C2}s_{E2}s_{F1}(95, 90)$

already recorded at the plan paths graph's nodes as a result of the selection process, without re-computing these instances from scratch. This requires the optimal instances of the nodes to remain valid prior to change occurrence which, however, is not the case during execution. More specifically, once a service $s \in candidates(t)$ (of the selected composite service) is invoked, the optimal instances recorded at the remaining non-executed nodes are no longer valid, since they consider other candidate services for task t's execution. For instance, in the example of Section 6.2, once the execution of service s_{B1} starts, the optimal instances recorded at node F (see Table 5.2) are no longer valid, and the new optimal instances of this node are shown in Table 6.5 (note that optimal instance $s_{B1}s_{C1}s_{E1}s_{F1}$ was not previously regarded as optimal due to being dominated by instance $s_{B2}s_{C1}s_{E2}s_{F1}$, which is currently not valid).

To tackle this, we modify the selection process such that the selection algorithm is applied on the *reverse* version of the plan paths graph, generated by reversing the direction of edges in the original plan paths graph (i.e. the start node of the reverse plan paths graph is the end node of the original one). For example, the reverse graph for the plan paths graph of Figure 4.2 is provided in Figure 6.1. Such modified selection produces the same optimal composite solution, while improving the performance of adaptation to changes at execution time. This is because, when performing the selection algorithm on the reverse plan paths graph, each node t stores the optimal instances from the destination of the original graph (the start node of the reverse graph) to t, and hence executing a component service of the selected solution only affects the

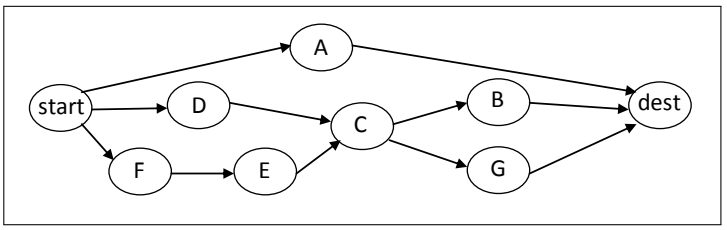

FIGURE 6.1: Reverse plan paths graph for *plan holiday* task

optimal instances of the corresponding node, without having any impact on the other, non-executed nodes.

To illustrate, consider the reverse graph of Figure 6.1, and assume the optimal solution generated by the selection process is $s_B s_C s_E s_F$. Since the optimal instances recorded at node C after selection are instances of paths DC and FEC, they remain valid after the execution of service s_B. The same holds for the optimal instances of nodes D, E and F. In contrast, when performing selection using the original graph (Figure 4.2), executing service s_B affects the optimal instances of nodes C, D, E, F and *dest*, which, in this case, maintain instances of paths containing node B. It is worth noting here that, with the reverse approach, responding to a deviation in the quality values delivered by the invoked service s_B only involves updating the optimal instances of node B, as opposed to the original approach, which requires modifying the optimal instances of nodes C, D, E, F and *dest* in order to obtain the new optimal selection. In other words, the reverse approach considerably lessens the delay time required for re-selection when the qualities of a selected service deviate from that expected, and this deviation cannot be anticipated prior to completing the execution of the violating service. Such a case can occur, for instance, due to network load, or when the service provider advertises false qualities to attract more customers.

The reactive service execution algorithm based on the reverse graph approach is presented next.

6.3.2 Reactive execution algorithm

To facilitate execution-time reactivity, the reverse plan paths graph (V_{RPK}, E_{RPK}), produced by the selection process, is kept valid by adjusting it proactively each time a new component service of the selected optimal solution is invoked. This simply involves updating the graph nodes' valid predecessors, and changing the destination node (of the reverse graph), as follows. Consider a selected composite service $ins_{ex} + s_{inv} + ins_{uex}$, where instance $ins_{ex} \in instances(p_{ex})$ is already executed and service $s_{inv} \in candidates(t_{inv})$ has just been invoked. Here, p_{ex} and t_{inv} are the already executed sub-plan and the invoked task, respectively, while ins_{uex} is the remaining, not yet executed instance. The set of plans acceptable according to plan-based pruning $SPLAN$ (with respect to which the valid predecessors are computed) should thus be adjusted so that only those plans beginning with tasks $p_{ex} + t_{inv}$ are kept in $SPLAN$, while all other plans are removed due to becoming invalid from this point. That is, $SPLAN := SPLAN \cap \{p \in plan(task_r) \mid beginwith(p, p_{ex} + t_{inv})\}$, where function $beginwith(p, p_{ex} + t_{inv})$ returns true if plan p starts with sub-plan $p_{ex} + t_{inv}$. Moreover, the task being executed t_{inv} is considered the new destination node of the reverse graph, with its request-based non-dominated services being set to instance $ins_{ex} + s_{inv}$, i.e. $rcandidates(t_{inv}) = \{ins_{ex} + s_{inv}\}$, and its valid predecessors being assigned an unprocessed status so that their optimal instances are recalculated when a change occurs, i.e. $\forall p \in validpred(t_{inv}), \quad status(t_{inv}, p) = \text{unp}$. Note that the assumption here is that once a service starts execution, it completes its execution successfully without exceptions.

Now, given the adjusted reverse plan paths graph, responding to a change at node t_{ch} while executing service s_{inv}, is achieved by applying the same steps performed when reacting to a change during selection. In other words, the request-based non-dominated services of node t_{ch} are updated, the changes required to the affected optimal instances

are identified, the valid predecessors of the nodes are updated, and the node to which the selection algorithm should roll back is identified. Starting from the roll back point, the selection algorithm continues its execution in a *reactive* manner (to address changes that might occur during the re-selection process) until reaching the end node (task t_{inv}), which stores the new optimal solution (the one with the highest utility among the optimal instances recorded at node t_{inv}). This reactive re-selection is provided in Algorithm 8.

Algorithm 8 R-ReSelection-Algorithm $(t_{inv}, g_{RPK}(V_{RPK}, E_{RPK}))$

```
1: endNodeInd ← the index of t_inv in the topological order of g_RPK
2: nextInd ← endNodeInd
3: while true do
4:    observe the world
5:    if change occurred then
6:       process-changes(nextInd)
7:    while nextInd < endNodeInd do
8:       v ← node at position nextInd according to topological order
9:       currInd=nextInd
10:      nextInd=nextInd+1
11:      if v = v_start or validpred(v) ≠ ∅ then
12:         E_v = {(v, u) ∈ E_PK}
13:         while (not empty(E_v)) and (nextInd > currInd) do
14:            (v, u) ← an element from E_v
15:            E_v ← E_v \ {(v, u)}
16:            r-process-edge(v,u,currInd,nextInd)
17:   Solution ins_sol is the optimal instance at t_inv with the highest utility_c
```

6.3.3 Example

Consider the reverse plan paths graph of Figure 6.1, and assume the request-based non-dominated services of the nodes are as provided in Table 5.1, with the following modification: task G has another available candidate service, which is service $s_{G2}(5, 60)$. Given the same example request of Section 6.2 (the user wants to minimise price with the constraint that the execution time should be less than 100), the set of plans to be considered for composition according to plan-based pruning are

TABLE 6.6: The valid predecessors for the nodes of Figure 6.1 in the running example

Node	A	B	C	D	E	F	G
validpred	$\emptyset$	$SFEC$	SFE	$\emptyset$	SF	S	$SFEC$

TABLE 6.7: Optimal instances recorded at the reverse graph's nodes as a result of the selection process in the running example

Node	F	E	C	B	G
validpred	S	SF	SFE	$SFEC$	$SFEC$
oinstances (ex,pr)	$s_{F1}(30,10)$ $s_{F2}(20,40)$	$s_{F1}s_{E1}(57,15)$ $s_{F1}s_{E2}(45,30)$ $s_{F2}s_{E2}(35,60)$	$s_{F1}s_{E1}s_{C1}(72,65)$ $s_{F1}s_{E2}s_{C1}(60,80)$ $s_{F2}s_{E2}s_{C1}(50,110)$ $s_{F1}s_{E1}s_{C2}(87,45)$ $s_{F1}s_{E2}s_{C2}(75,60)$	$s_{F1}s_{E2}s_{C1}s_{B1}(80,110)$ $s_{F2}s_{E2}s_{C1}s_{B1}(70,140)$ $s_{F1}s_{E2}s_{C2}s_{B1}(95,90)$ $s_{F1}s_{E2}s_{C1}s_{B2}(90,92)$	$s_{F1}s_{E1}s_{C1}s_{G2}(77,125)$ $s_{F1}s_{E2}s_{C1}s_{G2}(65,140)$ $s_{F2}s_{E2}s_{C1}s_{G2}(55,170)$ $s_{F1}s_{E1}s_{C2}s_{G2}(92,105)$ $s_{F1}s_{E2}s_{C2}s_{G2}(80,120)$

TABLE 6.8: The valid predecessors for the nodes of Figure 6.1 after executing service s_{B1} in the running example

Node	A	B	C	D	E	F	G
validpred	$\emptyset$	$SFEC$	SFE	$\emptyset$	SF	S	$\emptyset$

$SPLAN = \{BCEF, GCEF\}$. Based on this, the valid predecessors of the nodes in the reverse graph, and their optimal instances are as shown in Tables 6.6 and 6.7, respectively. As can be seen, the cheapest satisfactory composite service is service $s_{B1}s_{C2}s_{E2}s_{F1}$.

After invoking service s_{B1}, set $SPLAN$ is adjusted by removing all the abstract plans not including task B (i.e. $SPLAN := SPLAN \setminus \{GCEF\}$), resulting in a corresponding update in the valid predecessors of the nodes, as provided in Table 6.8. In addition, node B is now regarded as the new destination node, with $rcandidates(B) = \{s_{B1}\}$, and its valid predecessor is assigned an unprocessed status (the optimal instances recorded at this node are no longer valid, and thus require recalculation).

Now, suppose that during the execution of service s_{B1}, service $s_{C3}(20, 35)$ joins the candidate services of node C. To tackle this change, the selection algorithm execution rolls back to node E so that the optimal instances of nodes C and B are modified (this modification is performed concurrently with the execution of service s_{B1}). Table 6.9

TABLE 6.9: Optimal instances recorded at nodes C and B in response to the addition of service s_{C3} in the running example

Node	C	B
validpred	SFE	$SFEC$
oinstances (ex,pr)	$s_{F1}s_{E2}s_{C1}(60, 80)$	$s_{F1}s_{E2}s_{C1}s_{B1}(80, 110)$
	$s_{F2}s_{E2}s_{C1}(50, 110)$	$s_{F2}s_{E2}s_{C1}s_{B1}(70, 140)$
	$s_{F1}s_{E1}s_{C2}(87, 45)$	$s_{F1}s_{E2}s_{C2}s_{B1}(95, 90)$
	$s_{F1}s_{E2}s_{C2}(75, 60)$	$s_{F1}s_{E1}s_{C3}s_{B1}(97, 80)$
	$s_{F1}s_{E1}s_{C3}(77, 50)$	$s_{F1}s_{E2}s_{C3}s_{B1}(85, 95)$
	$s_{F1}s_{E2}s_{C3}(65, 65)$	$s_{F2}s_{E2}s_{C3}s_{B1}(75, 125)$
	$s_{F2}s_{E2}s_{C3}(55, 95)$	

shows the new optimal instances of nodes C and B, of which, instance $s_{B1}s_{C3}s_{E1}s_{F1}$ recorded at the destination node B is the new optimal solution.

6.4 Change Categories at Execution Time

Changes occurring at execution time can be divided into two categories (see Figure 6.2): changes not to be considered and changes to be considered. We assume that the time required to identify the change category is negligible (especially when compared to re-selection time), and therefore is ignored in the following detailed categorisation of changes.

6.4.1 Changes not to be considered

A change in the available services of task t_{ch} while executing task t_{inv} need not be considered (i.e. does not trigger the re-selection process) **iff** *one* of the following is satisfied:

- task t_{ch} is already executed, i.e. $t_{ch} = t_{inv}$ or t_{ch} appears *after* t_{inv} according to the topological order of the reverse graph;

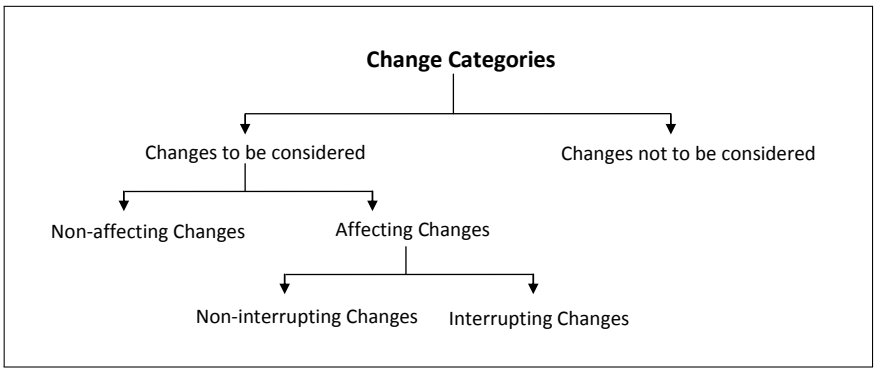

FIGURE 6.2: Change categories at execution time

- task t_{ch} is not part of the plan being executed (p_{sel}) and does not belong to any satisfactory plan after the change, i.e. $(t_{ch} \notin nodes(p_{sel})) \wedge (validpred_n(t_{ch}) = \emptyset)$; or
- the request-based non-dominated services of task t_{ch} are not affected by the change, i.e. $ADD = RMV = \emptyset$.

6.4.2 Changes to be considered

A change in the available services of task t_{ch} while executing task t_{inv} need to be considered (i.e. trigger the re-selection process) **iff** *all* of the following are satisfied:

- task t_{ch} is not executed yet, i.e. t_{ch} appears *before* t_{inv} according to the topological order in the reverse graph;
- task t_{ch} is part of the plan being executed (p_{sel}) or belongs to at least one satisfactory plan after the change, i.e. $(t_{ch} \in nodes(p_{sel})) \vee (validpred_n(t_{ch}) \neq \emptyset)$; and
- the request-based non-dominated services of task t_{ch} are affected by the change, i.e. $(ADD \neq \emptyset) \vee (RMV \neq \emptyset)$.

Changes to be considered are further divided into non-affecting changes and affecting changes, as detailed next.

6.4.2.1 Non-affecting changes

Non-affecting changes are the changes that do not affect the optimal composite service being executed, but should be reflected in the optimal instances recorded at nodes so that these instances remain valid regarding the current state of the environment. Having no impact on the optimal solution, this category of change does not cause any delay to the execution process. In other words, the composite service can continue its execution even if the adaptation to the change is still running. Generally, a change to be considered is regarded as non-affecting in the following cases: the deletion of a non-selected service (a service that is not part of the current optimal solution); and changes in the quality values of a non-selected service s_o (s_{ch} denotes this service after the change) such that (s_o r-dm s_{ch}) $\vee$ ((s_o is incomparable with s_{ch}) $\wedge$ ($s_{ch} \notin ADD$)).

6.4.2.2 Affecting changes

Affecting changes are the changes that might cause a modification to the optimal composite service being executed, and are divided into non-interrupting changes and interrupting changes.

Non-interrupting changes are those affecting changes, the reaction to which does not cause any interruption between service executions, since the next service to be executed can be identified without the need for re-selection to be completed. Specifically, an affecting change to the services of task t_{ch} is non-interrupting **iff** task t_{ch} is the next task to be executed according to the current optimal solution, with service s_{sel} being the currently selected service for this task, and *one* of the following is satisfied:

- the change is the addition of a new service s_n such that s_n r-dm s_{sel}; or
- the change is a modification in the quality values of service s_o (s_{ch} denotes this service after the change) such that s_{ch} r-dm s_{sel} (note that s_{sel} might be the service affected by the change, i.e. $s_o = s_{sel}$).

Intuitively, responding to such changes will result in replacing service s_{sel} with service s_n (in the addition case), and with service s_{ch} (in the modification case). Hence, the next service can be anticipated without requiring interruption.

Interrupting changes are those affecting changes, the reaction to which might result in an interruption to the composite service execution. This is because the next service to be executed cannot be identified prior to performing re-selection, thus causing the execution process to stop until re-selection is completed. Specifically, an affecting change to the services of task t_{ch} is interrupting in the following cases.

Case 1: the addition of a new service s_n such that *one* of following is satisfied:

- t_{ch} is not part of the plan being executed;
- t_{ch} belongs to the plan being executed and s_n is incomparable with s_{sel} (the currently selected service for task t_{ch}); or
- t_{ch} belongs to the plan being executed, but is not the next task in the execution sequence, and s_n r-dm s_{sel}.

Case 2: the deletion of a selected service (a service that is part of the currently selected solution).

Case 3: changes in the quality values of a non-selected service s_o (s_{ch} denotes this service after the change) such that *all* of the following are satisfied:

- (s_{ch} r-dm s_o) $\vee$ ((s_{ch} is incomparable with s_o) $\wedge$ ($s_{ch} \in ADD$)); and
- (t_{ch} is not the next task to be executed) $\vee$ ($\neg$(s_{ch} r-dm s_{sel})), where s_{sel} is the currently selected service for task t_{ch}.

Case 4: changes in the quality values of a selected service s_{sel} (s_{ch} denotes this service after the change) such that t_{ch} is not the next task to be executed or $\neg$(s_{ch} r-dm s_{sel}).

6.5 Reactive System Behaviour

Based on the above change categories, the reactive behaviour of the system during execution can be modelled using the finite state automaton provided in Figure 6.3, which consists of five states. The states prefixed with ex, *ex-* α, indicate that a component service of the optimal solution is currently running and, at the same time, the following are satisfied according to the value of α:

- when α = *nch*, no re-selection is being performed by the composite service provider;
- when α = *naff*, a re-selection is being performed by the composite service provider in response to a set of non-affecting changes;
- when α = *nint*, a re-selection is being performed by the composite service provider in response to a set of changes including at least one non-interrupting change and no interrupting changes; and
- when α = *int*, a re-selection is being performed by the composite service provider in response to a set of changes including at least one interrupting change.

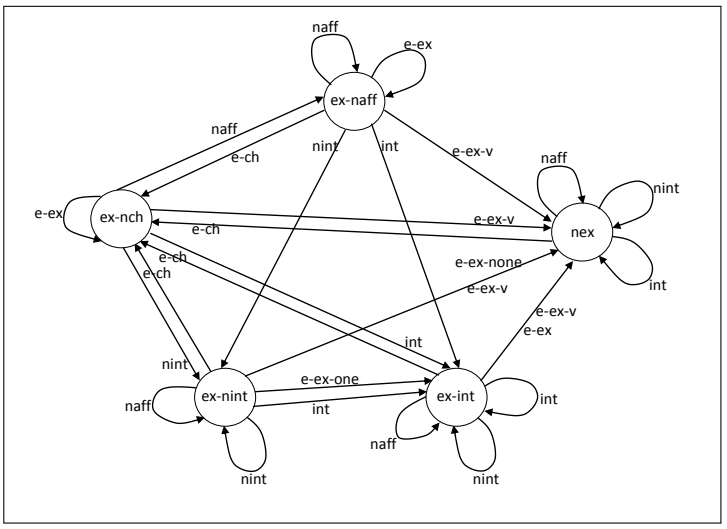

FIGURE 6.3: The finite state machine modelling reactive behaviour during execution

State *nex* indicates that the optimal solution execution is currently interrupted until re-selection by the composite service provider is completed.

State transitions can be triggered by the following events:

- event *naff*, indicating the occurrence of a non-affecting change;
- event *nint*, indicating the occurrence of an affecting, non-interrupting change;
- event *int*, indicating the occurrence of an affecting, interrupting change;
- event *e-ch*, indicating the end of the change adaptation process (i.e. service re-selection is completed). As a result of this, the currently selected optimal solution might be updated, and future change categories will be identified with respect to this new solution;
- event *e-ex*, indicating the end of the current component service execution, with this component delivering its expected qualities;
- event *e-ex-v*, indicating the end of the current component service execution, with this component violating its expected qualities;

- event *e-ex-one*, indicating the end of the current component service execution, such that this component delivers its expected qualities, and only one service, in the set of the request-based non-dominated services of task t_{nxt} (the next task in the execution order), request-based dominates the currently selected service s_{nx-sl} for this task, i.e., $|dominating_s(s_{nx-sl})| = 1$; and

- event *e-ex-none*, indicating the end of the current component service execution, such that this component delivers its expected qualities, and the number of services, in the set of the request-based non-dominated services of task t_{nxt} (the next task in the execution order), that request-based dominate the currently selected service s_{nx-sl} for this task, is not 1, i.e., $|dominating_s(s_{nx-sl})| \neq 1$.

According to Figure 6.3, the behaviour of the system is interpreted as follows. The reactive execution begins in state *ex-nch*, by invoking the first component service. With the occurrence of a change to be considered during a component execution, the system transitions to one of the states *ex-naff*, *ex-nint*, or *ex-int*, based on the change category, identified with respect to the currently selected solution. This solution may be updated each time a re-selection is completed (event *e-ch*), causing the system to return to state *ex-nch*. Note that re-selection may be initiated several times during a component execution. After the successful execution of a component service (event *e-ex*), the state into which the system transitions is determined based on its current state, as follows.

If the system is in state *ex-nch* (i.e. no re-selection is being performed), the next service in the currently selected solution is invoked, without changing the state of the system.

If the system is in state *ex-naff* (i.e. the re-selection being performed will not affect the currently selected solution), the next service in this solution is invoked, without

changing the state of the system. In other words, the re-selection is carried on while executing the next service.

If the system is in state *ex-int* (i.e. the next service to be executed cannot be identified before completing the re-selection being performed), the system transitions to state *nex*, and remains in this state until re-selection is completed and the next service to be invoked is determined.

If the system is in state *ex-nint*, the following two cases are distinguished. Case 1: $|dominating_s(s_{nx-sl})| = 1$, in which the next service to be executed can be estimated without the need for re-selection to be completed. This service, service $s_{nx-new} \in dominating_s(s_{nx-sl})$, is thus invoked without delaying execution, causing the system to transition to state *ex-int*. In other words, the re-selection is continued while executing service s_{nx-new}, but is considered interrupting since the next service to invoke after service s_{nx-new} cannot be known prior to completing re-selection. Case 2: $|dominating_s(s_{nx-sl})| \neq 1$, in which the next service to be executed cannot be determined before the re-selection is completed. Therefore, the execution process is interrupted by moving to state *nex* in order to continue the re-selection.

The case where the current component service delivers unexpected quality values (event *e-ex-v*) is considered an interrupting change, and thus also causes the system to enter state *nex*, regardless of its current state.

Example. Consider again the example of Section 6.3.3, where composite service $s_{B1}s_{C2}s_{E2}s_{F1}$ is the initially selected solution. Invoking the first component service, s_{B1}, initiates the reactive execution-time behaviour at state *ex-nch*. Since the addition of service s_{C3} while executing service s_{B1} is an interrupting change (see Section 6.4.2.2), it triggers the transition of the system to state *ex-int* (see Figure 6.4(a)) to indicate a running re-selection process. Once the re-selection is completed, the system goes back to state *ex-nch* (see Figure 6.4(b)), updating the optimal solution to $s_{B1}s_{C3}s_{E1}s_{F1}$.

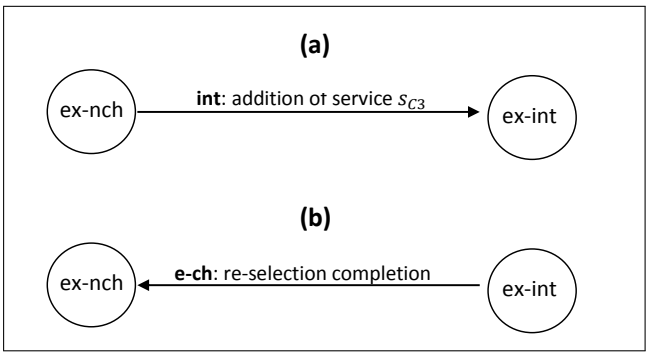

FIGURE 6.4: System transitions corresponding to the example of Section 6.3.3

6.6 Analytical Study

The goal of this section is to analyse the time complexity of the proposed reverse re-selection algorithm, and to compare it against reselection from scratch. The latter is assumed also to be applied to the reverse version of the plan paths graph for ease of comparison. Like the reactive selection analysis, the focus here is on the step of optimal instances modification in response to a change since, compared to this step, the time required for the other change handling steps (e.g. updating the request-based non-dominated services and the status of affected nodes) is negligible.

Since the valid predecessors at each node are processed independently, the analysis assumes for simplicity one valid predecessor per node (the specific case of one abstract plan). This still allows us to demonstrate the efficiency gain achieved by our approach for any affected valid predecessor, and can be easily generalised to handle the case of multiple valid predecessors (i.e. multiple abstract plans). Note that, in such a general case, our approach achieves further time reduction due to reprocessing only *affected* valid predecessors per each node, as opposed to reselection from scratch, which reprocesses *all* the valid predecessors.

6.6.1 Reselection from scratch

The reselection-from-scratch approach recalculates the optimal instances of all non-executed nodes from scratch, in response to a change to be considered at node v_{ch} while executing node v_{inv}. Thus, its time complexity, $\tau^{srch}(resel)$, is:

$$\tau^{srch}(resel) = \sum_{i=2}^{inv-1} \tau^{srch}(oinstances_n(v_i)) \tag{6.1}$$

Here, $\tau^{srch}(oinstances_n(v_i)$ is the time required for *recomputing* the optimal instances at node v_i, and is given as follows (corresponding to Equations 4.5 and 5.3, respectively):

$$\tau^{srch}(oinstances_n(v_{i<ch})) \text{ is } O(n^{2i} \times \prod_{m=1}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2) \tag{6.2}$$

$$\tau^{srch}(oinstances_n(v_{i\geq ch})) \text{ is } O(n^{2(i-1)} \times n'^2 \times \prod_{m=1}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2) \tag{6.3}$$

where $n' = |candidates_n(v_{ch})|$, i.e. $n' = n + 1$ in case of service addition; $n' = n - 1$ in case of service deletion; and $n' = n$ in case of changes in service qualities.

6.6.2 Reverse Reselection

The reverse-reselection (the proposed) approach only makes the updates necessary to the affected optimal instances, in response to a change to be considered at node v_{ch} while executing node v_{inv}, without recalculating those instances from scratch. Thus, its time complexity, $\tau^{r}(resel)$, is:

$$\tau^{r}(resel) = \sum_{i=ch}^{inv-1} \tau^{r}(oinstances_n(v_i)) \tag{6.4}$$

Here, $\tau^{r}(oinstances_n(v_i))$ is the time required for *modifying* the optimal instances at node v_i. The modification depends on the type of change that has occurred, and can

be analysed similarly to Section 5.4.2. That is, for the addition case:

$$\tau^r(oinstances_n(v_{ch})) \text{ is } O(n^{2ch-1} \times \prod_{m=1}^{ch-1} sr_m^2 \times \prod_{m=2}^{ch-1} ir_m^2 \times sr_{ch} \times ir_{ch})$$

$$\tau^r(oinstances_n(v_{i>ch})) \text{ is } O(n^{2(i-1)} \times n' \times \prod_{\substack{m=1 \\ m \neq ch}}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2 \times sr_{ch} \times ir_i)$$

Similarly, for the deletion case:

$$\tau^r(oinstances_n(v_{i \geq ch})) \text{ is } O(2 \times n^{2i-2} \times (n-1) \times \prod_{\substack{m=1 \\ m \neq ch}}^{i} sr_m^2 \times \prod_{m=2}^{i-1} ir_m^2 \times sr_{ch} \times ir_i)$$

It is important to note that complexity in Equation 6.4 is applicable as long as $ch < inv$ (the node affected by the change is not the node being executed). Yet, when $ch = inv$ (the invoked service delivers unexpected quality values), reselection involves only recombining the optimal instances already recorded at node v_{inv-1} with the modified invoked instance. That is:

$$\begin{aligned} \tau^r(resel) &\text{ is } O(|oinstances_o(v_{inv-1})|) \\ &\text{is } O(n^{inv-1} \times \prod_{m=1}^{inv-1} sr_m \times \prod_{m=2}^{inv-1} ir_m) \end{aligned} \tag{6.5}$$

(see Equation 4.4)

6.6.3 Comparison

To compare time complexities τ^{srch} and τ^r, let's make again the simplifying assumption that $\forall i,\ sr_i = ir_i = r$. It can be easily concluded, similarly to Section 5.4.3, that when the change occurs at a non-executed node (i.e. $ch < inv$), the proposed reverse re-selection approach reduces the processing time at each affected node by a factor of n.

Moreover, the reduction factor increases further in the case where the change affects the node being executed (i.e. $ch = inv$), and is given by the following two lemmas.

Lemma 6.1. *When $r^2 > \frac{1}{n}$ and $ch = inv$, the proposed reverse approach reduces re-selection time by a factor of $n^{inv-1} \times r^{2inv-5}$*

Proof. Since, from Equations 6.2 and 6.3, $\tau^{srch}(oinstances_n(v_i))$ is $O(n^{2i} \times r^{4i-4})$, the following holds if $r^2 > \frac{1}{n}$:

$$\forall i \in [2, inv-2],\ \tau^{srch}(oinstances_n(v_i)) < \tau^{srch}(oinstances_n(v_{i+1}))$$

From Equation 6.1, and assuming inv is a small number, i.e. $inv \leq k << n$, we can conclude that:

$$\tau^{srch}(resel) \text{ is } O(\tau^{srch}(oinstances_n(v_{inv-1})))$$
$$\text{is } O(n^{2inv-2} \times r^{4inv-8})$$

As a result, comparing time complexities τ^{srch} and τ^r (see Equation 6.5), leads to:

$$\frac{\tau^{srch}(resel)}{\tau^r(resel)} = \frac{n^{2inv-2} \times r^{4inv-8}}{n^{inv-1} \times r^{2inv-3}} = n^{inv-1} \times r^{2inv-5}$$

□

Lemma 6.2. *When $r^2 \leq \frac{1}{n}$ and $ch = inv$, the proposed reverse approach reduces re-selection time by a factor of $n^{5-inv} \times r^{7-2inv}$*

Proof. Since, from Equation 6.2 and 6.3, $\tau^{srch}(oinstances_n(v_i))$ is $O(n^{2i} \times r^{4i-4})$, the following holds if $r^2 \leq \frac{1}{n}$:

$$\forall i \in [2, inv-2],\ \tau^{srch}(oinstances_n(v_i)) \geq \tau^{srch}(oinstances_n(v_{i+1}))$$

From Equation 6.1, and assuming inv is a small number, i.e. $inv \leq k << n$, we can conclude that:

$$\tau^{srch}(resel) \text{ is } O(\tau^{srch}(oinstances_n(v_2)))$$
$$\text{is } O(n^4 \times r^4)$$

As a result, comparing time complexities τ^{srch} and τ^r (see Equation 6.5), leads to:

$$\frac{\tau^{srch}(resel)}{\tau^r(resel)} = \frac{n^4 \times r^4}{n^{inv-1} \times r^{2inv-3}} = n^{5-inv} \times r^{7-2inv}$$

□

6.7 Empirical Study

The goal of this section is to assess the efficiency of our reverse graph re-selection, the gain in utility by responding to changes ahead of time, and the reduction in the delay during execution as a result of performing re-selection without interfering with the execution process (unless necessary).

To evaluate the gain in performance obtained by the reverse approach, the time required for re-selecting services for the remaining tasks, in response to an affecting change at execution time, is compared in two cases: where *forward* selection is applied, and where *reverse* selection is applied. As explained in Section 6.3, reacting to a change in

the former case requires recalculating the optimal instances of the non-executed nodes from scratch, while in the latter case, the adaptation process is achieved by making only the necessary updates to the optimal instances already recorded at nodes. Here, the number of candidate services per task is fixed at 500, while the execution position (the index of the service being executed) when the change occurs is varied between 1 and 9 (10 is the total number of services in the selected solution). Figures 6.5(a) and 6.5(b) compare the two cases in terms of running time, averaged over a number of different random requests, and a number of randomly generated graph instances, each containing 10 tasks per abstract plan. In Figure 6.5(a), change types (addition, deletion, or changes in qualities) and locations (the tasks and services affected) at each execution position are selected randomly, whereas those considered in Figure 6.5(b) are receiving unexpected quality values from the executed services. As can be seen, the reverse re-selection significantly outperforms the forward re-selection, especially when the change is discovered at an early stage of execution. Moreover, both cases require less time with the increasing execution position. This is because, as more services are executed, the number of graph nodes to be considered in the re-selection process decreases, and so does the number of their optimal instances. We can also observe from the situation studied in Figure 6.5(b), in which it is not possible to perform the adaptation process in parallel with execution since the erroneous behaviour cannot be discovered prior to its occurrence, that almost no interruption in execution will be caused by the reverse approach proposed.

To evaluate utility gain from the early reaction to changes, the solution optimality obtained, as a result of reacting to a change in the selected services as soon as it occurs in the environment, is compared with that of the delayed reaction (i.e. when the unavailable service is invoked or after the quality violating service is executed). This optimality is estimated as $\frac{utility_{c_{act}}}{utility_{c_{opt}}}$, where $utility_{c_{act}}$ is the actual utility achieved by re-selecting services for the non-executed tasks, and $utility_{c_{opt}}$ is the optimal utility

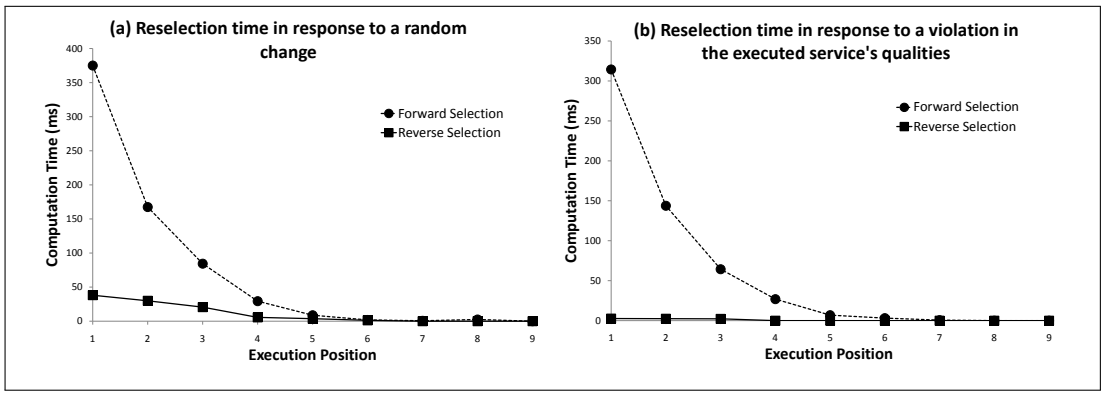

FIGURE 6.5: Reverse re-selection vs. forward re-selection in terms of computation time

assuming no task is executed (i.e. the utility of the best solution according to the current environment state, and given that no task is executed). Figures 6.6(a) and 6.6(b) show the results (averaged over a number of requests and graph instances) in the cases where the last service in the selected solution becomes unavailable, and changes its qualities, respectively, varying the execution position at which the change occurs (i.e. at which the re-selection is triggered by the early approach to change handling) between 1 and 9 (each solution composite service comprises 10 services). As expected, the earlier change adaptation is performed, the better the utility of the resulting solution, which emphasises the importance of responding to changes as early as possible. This is further highlighted in Figures 6.6(c) and 6.6(d), where the execution position at which the change is observed is fixed at 1 (i.e. the change occurs while executing the first service of the selected solution), while the change location (the index of the task affected by the change) ranges between 2 and 10. Clearly, the optimality achieved by the delayed re-selection decreases as more services become executed.

Finally, the last part of the experiments evaluates the reduction in interruption time between component service executions, achieved as a result of reacting to changes in the environment as soon as they occur, in parallel with the execution of the current component service. This parallelism is simulated using multi-threading on the

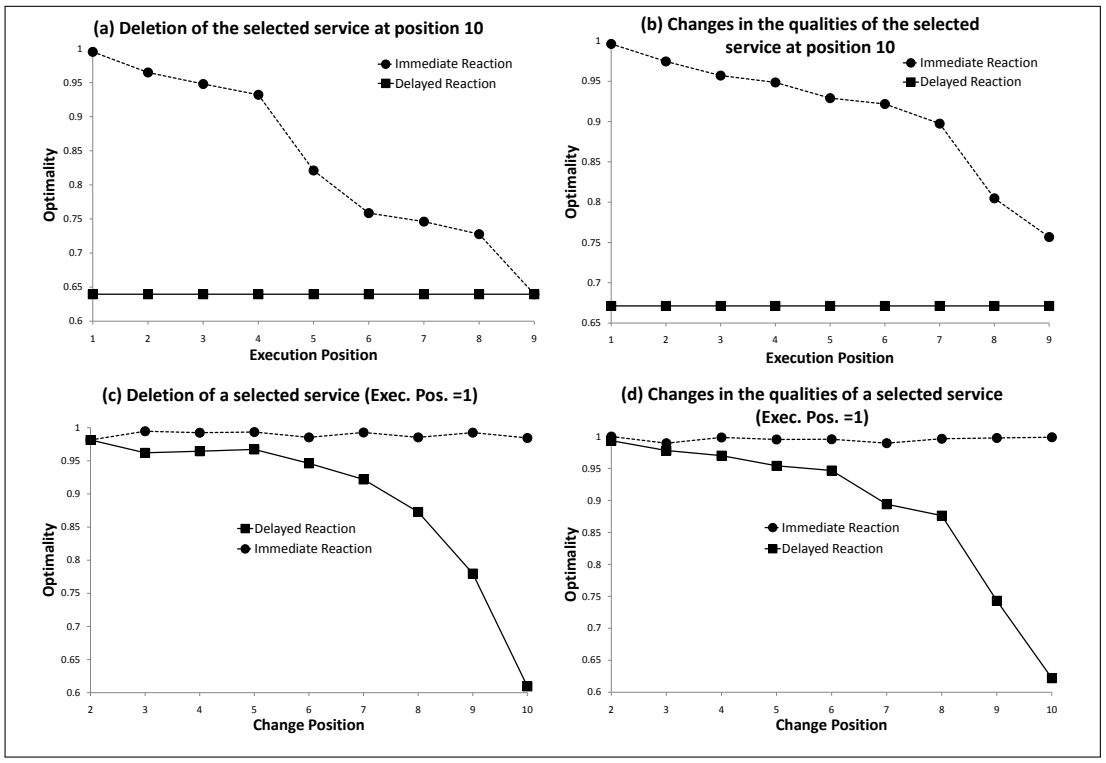

FIGURE 6.6: Evaluating the gain in utility

composite service provider side, with the execution of a component service being simulated by invoking a service on a remote computer, which simply sleeps for a certain amount of time $srvExecTime$ (service execution time) before returning a result. Changes occurring during execution are generated randomly in the interval $[start, end = start + (srvExecTime * maxSrvNum)]$, where $start$ is the start time of the composite service execution, while $maxSrvNum$ is the maximum number of component services in a composite solution. The type of each change (addition, deletion, or changes in qualities), and its location (the task and the service affected by the change) are also selected randomly. Figure 6.7 shows the delay time after completing the execution of each component service in the composite solution, averaged over a number of different runs. In each run, 20 changes to be considered are introduced during each component service execution, while the number of candidate services per

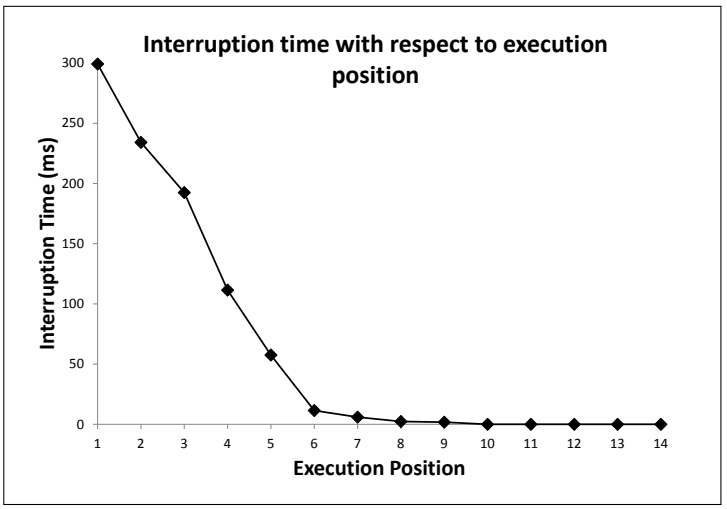

FIGURE 6.7: Interruption time versus execution position

task, service execution time $srvExecTime$, and the number of tasks per abstract plan $maxSrvNum$, are fixed at 500, 5 seconds, and 15 tasks, respectively. The results indicate that, even with this large number of changes, the interruption time achieved by our approach is small (i.e. the re-selection process in response to the changes is almost completed before the current component service finishes its execution), especially with the increasing execution position (the re-selection process requires less time when more component services become executed, due to the decreased number of nodes to be considered in the re-selection process).

The interruption time is further evaluated in Figures 6.8(a) and 6.8(b) with respect to service execution time $srvExecTime$. Figure 6.8(a) shows the interruption time (averaged over a number of runs) between the first and second component service executions, varying the number of changes introduced during the first service execution between 15 and 55 (all the changes are assumed to be interrupting). As expected, the interruption time decreases with the decreasing number of changes and the increasing service execution time. Figure 6.8(b), on the other hand, shows the effect of varying the time slot within which the change occurs on interruption time. For this purpose,

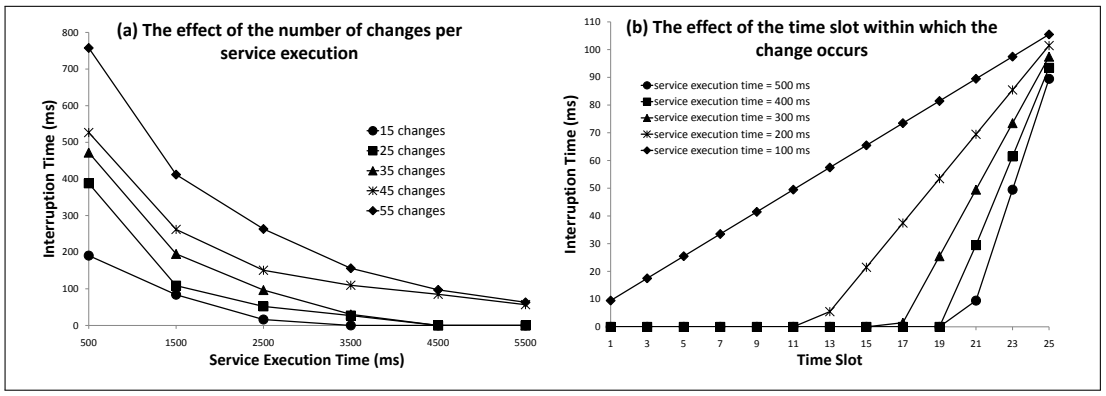

FIGURE 6.8: Interruption time with respect to service execution time

the service execution time $srvExecTime$ is divided into 25 equal time intervals, during which an interrupting change is introduced while executing the first component service. Intuitively (as shown in Figure 6.8(b)), the earlier the change occurs during a service execution, the more likely that no interruption will be caused by the corresponding re-selection.

6.8 Conclusion

In this chapter, we have presented a novel reactive execution algorithm, which adapts to execution-time service changes for both repairing and optimisation purposes. To achieve a light adaptation process in response to changes, the algorithm reuses the optimal instances generated during the selection process. For this purpose, it assumes a reverse version of the search graph, which allows response to changes by applying only a minimal number of modifications to the graph, without the need to perform re-selection from scratch. The adaptation process is triggered as soon as changes occur in the environment, without interfering with the execution process, unless necessary. This need is identified based on a categorisation of changes, specifying their urgency and importance, and guiding the behaviour of the executing system. Via such an

early, parallel-to-execution, and light reaction approach, the chances of a successful recovery are maximised and solution optimality is increased, while reducing execution disruptions as much as possible, as demonstrated through the evaluation conducted. The results also show that, even in the cases where interference with execution is non-preventable (e.g., when an executed service delivers unanticipated quality values), the algorithm manages to recover from the situation with minimal interruption.

The main priority of the proposed execution approach is increasing the satisfaction of the end-user (customer), by improving the quality of the selected solution, and eliminating any inefficiencies from the customer perspective (i.e. reducing the waiting time from execution interruption), even if this incurs some additional overhead on the composite service provider side. In other words, depending on the environment dynamism, the re-selection process may need to be instantiated several times by the composite service provider while a component service is executing, in order to keep the search graph valid with respect to the environment state, and consequently achieve an efficient adaptation in response to any change. In the case of component services with relatively short execution periods, such continuous light reaction seems both feasible and essential to ensure (or at least increase the chances) that any adaptation required terminates before the end of the current component's execution. Yet, such an approach may become unnecessarily costly and even impractical (from the composite service provider perspective) for long-running component services, since a valid instance of the search graph will have to be maintained for very long durations and the re-selection triggered a potentially very large number of times. More elaboration on the latter case, and an adjustment of the algorithm to handle it appropriately is provided when presenting the case study evaluation (see Section 8.6.4.1).

Chapter 7

Correlation-aware Service Selection

7.1 Introduction

Services are not independent from each other, with potential service correlations including flow correlation, compatibility correlation, and QoS correlation. In *flow correlation*, one service must be performed before another due to data or resource dependencies. This type of correlation is addressed in our model through the partial order relationship defined on sub-tasks in the planning knowledge hierarchy. *Compatibility correlation* concerns whether two services can be used together in a composition without integration errors. Such correlations are out of the scope of this thesis, but can be assessed, for example, by checking the semantic or syntactic compatibility of service interfaces. Here we focus on *QoS correlation*, a dependency between a service (the dependent) and a set of services (the dependees), regarding a quality attribute, and can be positive, where use of the dependees improves the attribute value for the dependent, or negative,

where it degrades the attribute value. For example, a flight booking service may charge an extra fee if payment is by credit card (negative correlation), but no fee with Visa Electron (positive correlation).

Accounting for quality correlations among services when performing service composition is essential to obtain more accurate quality estimations of service combinations, thus providing users with better solutions. Yet, most current composition approaches (including the approach presented so far in this thesis) fail to address such correlations by assuming independence between services regarding their quality values. In response, this chapter extends our previous selection model, by presenting a correlation-aware composition approach, where quality dependencies among services are modelled and considered during the selection process, while performing correlation-aware pruning prior to and during selection to improve performance.

The rest of the chapter is organised as follows. Section 7.2 presents a motivating example. The service model is adjusted in Section 7.3, in order to account for quality dependencies among services. Our correlation-aware search space reduction techniques are introduced in Section 7.4, followed by the correlation-aware selection algorithm in Section 7.5. The proposed algorithm is evaluated analytically and empirically in Sections 7.6 and 7.7, respectively. Finally, Section 7.8 concludes the chapter.

7.2 Motivating Example

Consider a user planning a holiday not wanting to exceed the price 700 for the whole travel plan while maximising reliability. The plan for achieving this goal comprises the tasks: *book flights*, *book hotel*, *book local transport*, and *book sightseeing*. Each sub-task has several candidate services, as in Figure 7.1, with price (pr) and reliability (rel) of these services in Figure 7.2. Now, suppose tourism company *trsmC* gives a special price

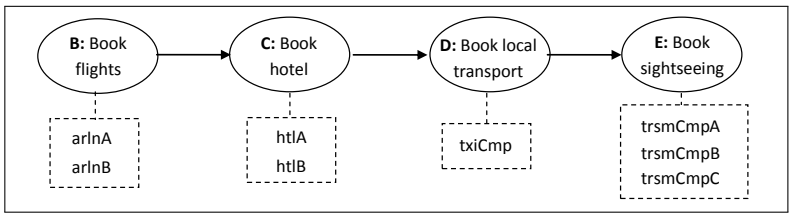

FIGURE 7.1: Plan for holiday and candidate services of sub-tasks

Service	reliability	price	Service	reliability	price
arlnA	95%	150	txi	95%	30
arlnB	90%	100	trsmA	92%	50
htlA	95%	470	trsmB	95%	70
htlB	95%	450	trsmC	97%	90

FIGURE 7.2: Quality values of candidate services in the running example

(50 instead of 90) if the hotel booked is *htlA*, due to an arrangement between them. If we assume independence between individual services (as in most existing approaches) and ignore the special price, we can select '*arlnA*, *htlB*, *txi*, *trsmB*' with values (rel:81%, pr:700) as the composition satisfying the price constraint with the highest reliability (the product of its components' reliabilities). But if we account for the special price, a better composition '*arlnA*, *htlA*, *txi*, *trsmC*' (rel:83%, pr:700), is selected.

Considering quality correlations in service selection is thus vital to find better solutions, but makes it difficult to prune uninteresting candidates prior to selection, since such pruning can disallow useful correlations. In our example, *htlA* is dominated by *htlB* since *htlB* offers a better price but the same reliability, so, if services were independent, *htlA* could be removed from the search without affecting the ability to find an optimal solution. However, due to the correlation, removing *htlA* disallows the optimal solution '*arlnA*, *htlA*, *txi*, *trsmC*'. Thus, despite being dominated, *htlA* should be kept, resulting in unnecessary consideration of four combinations (see Figure 7.3) for which replacing *htlA* with *htlB* gives a better price. With n tasks, m candidates per task, not eliminating a dominated service due to a correlation gives $(m-1)m^{n-2}$ combinations dominated by others. The same problem applies when pruning at sub-plan level. This requires

arlnA, htlA, txi, trsmA	arlnA, htlA, txi, trsmB
arlnB, htlA, txi, trsmA	arlnB, htlA, txi, trsmB

FIGURE 7.3: Service combinations dominated by other combinations

correlation-aware pruning.

7.3 Correlation-aware Service Model

Due to potential QoS correlations among services, the quality values delivered by a component service in a composition may vary according to the other component services selected. Hence, to address such dependencies between services, the attribute value function, $value_s$, of the service model (Section 3.4) is modified as follows:

$$value_s : S \times NAME_a \rightarrow 2^{VALUE_a \times DNF}$$

such that $\forall s \in S, \quad \forall a \in name_s(s)$,

$$value_s(s,a) = \{(v_1, c_1^\vee), (v_2, c_2^\vee), ..., (v_n, c_n^\vee)\} \subset dom_a(a) \times DNF$$

where $(v_i, c_i^\vee) \in value_s(s,a)$ indicates that service s delivers value v_i for attribute a if condition $c_i^\vee$ is satisfied. Here, each condition $c_i^\vee \in DNF$ is a logical formula in disjunctive normal form (where DNF is the set of all such formulas) restricting the services that can be selected with s in a composition, in order for v_i to be applicable. The propositional variables in these formulae are of the form $sel(t) \in S_t$, with function $sel(t)$ denoting the service selected for performing task t, and $S_t \subset candidates(t)$ being a set of services ($S_t \neq \emptyset$). Such propositional variables and their negations (i.e. the literals) have the following semantics: $sel(t) \in S_t$ indicates that service s should be used in a composition containing a service from set S_t, i.e. a composition performing task t and instantiating this task with a member of S_t; $\neg(sel(t) \in S_t)$ indicates that

service s should be used in a composition not containing a service from set S_t, i.e. either a composition not performing task t, or a composition mapping t to a service not in S_t. Since service s should not offer two different values for attribute a simultaneously, values $\{v_i\}_{1\leq i\leq n}$ are mutually exclusive, i.e. conditions $\{c_i^\vee\}_{1\leq i\leq n}$ are pairwise disjoint (if a particular service composition satisfies one of these conditions, the remaining conditions cannot be true).

Example. The price of service *trsmC*, in the scenario of Section 7.2, can be expressed as follows:

$$value_s(\text{trsmC}, \text{pr}) = \{(90, \neg(sel(\text{Book hotel}) \in \{\text{htlA}\})), \\ (50, sel(\text{Book hotel}) \in \{\text{htlA}\})\}$$

All other services have only one possible price, e.g. $value_s(\text{trsmA}, \text{pr}) = \{(50, \text{ALL})\}$, where $c^\vee = \text{ALL}$ indicates that price 50 is applicable in ALL compositions containing service *trsmA*, without restrictions.

Based on this, we introduce the following functions for later use (see Figure 7.4 for a summary of the notation used for the correlation-aware model).

- Function *values* returns all the values offered by a service for a quality attribute:

$$values : S \times NAME_a \rightarrow 2^{VALUE_a}$$

such that $\forall s \in S,\ \forall a \in name_s(s)$,

$$values(s, a) = \{v \in dom_a(a) \mid \exists c^\vee \in DNF,\ \ (v, c^\vee) \in value_s(s, a)\}$$

- Function *cond* returns the context condition associated with providing a particular value for a quality attribute by a service:

$$cond : S \times NAME_a \times VALUE_a \rightarrow DNF$$

such that $\forall s \in S,\ \forall a \in name_s(s),\ \forall (v_i, c_i^{\vee}) \in value_s(s,a),\ \ cond(s,a,v_i) = c_i^{\vee}$

- Function *cls* maps a condition in DNF to the set of conjunctive clauses comprising this condition:

$$cls : DNF \rightarrow 2^{DNF}$$

such that $\forall c^{\vee} = c_1^{\wedge} \vee c_2^{\wedge} ... \vee c_p^{\wedge} \in DNF,\ \ cls(c^{\vee}) = \{c_i^{\wedge}\}_{1 \leq i \leq p}$

- Function *ltr* maps a conjunctive clause to the set of literals comprising this clause:

$$ltr : DNF \rightarrow 2^{DNF}$$

such that $\forall c^{\wedge} = l_1 \wedge l_2 ... \wedge l_p \in DNF,\ \ ltr(c^{\wedge}) = \{l_i\}_{1 \leq i \leq p}$

- Function *ltask* returns the task that a literal is constraining:

$$ltask : DNF \rightarrow T$$

such that $ltask(sel(t) \in S_t) = t$, and $ltask(\neg(sel(t) \in S_t)) = t$.

- Function *lrange* returns the value range identified by a literal:

$$lrange : DNF \rightarrow 2^S$$

such that $lrange(sel(t) \in S_t) = S_t$, and $lrange(\neg(sel(t) \in S_t)) = S_t$

- Function *pltr* maps a conjunctive clause to the set of positive literals (i.e. propositional variables not proceeded by a negation) comprising this clause:

$$pltr : DNF \rightarrow 2^{DNF}$$

such that $\forall c^{\wedge} \in DNF, \quad pltr(c^{\wedge}) = \{l \in ltr(c^{\wedge}), \quad l \text{ is a positive literal}\}$

- Function *clst* returns the conjunctive clauses constraining a particular task, in a context condition:

$$clst : DNF \times T \rightarrow 2^{DNF}$$

such that $\forall c^{\vee} \in DNF, \ \forall t \in T,$

$$clst(c^{\vee}, t) = \{c^{\wedge} \in cls(c^{\vee}) \mid \exists l \in ltr(c^{\wedge}), \quad ltask(l) = t\}$$

- Function *task* returns the task that a service is performing:

$$task : S \rightarrow T$$

such that $\forall s \in S, \quad s \in candidates(task(s))$

- Function *addltr* adds literal $sel(task(s)) \in \{s\}$ to each conjunctive clause of a context condition:

$$addltr : DNF \times S \rightarrow DNF$$

such that $\forall c^{\vee} = c_1^{\wedge} \vee c_2^{\wedge} ... \vee c_p^{\wedge} \in DNF, \ \forall s \in S,$

$$addltr(c^{\vee}, s) = c_1^{\wedge\prime} \vee c_2^{\wedge\prime} ... \vee c_p^{\wedge\prime} \in DNF$$

where $ltr(c_i^{\wedge\prime}) = ltr(c_i^{\wedge}) \cup \{(sel(task(s)) \in \{s\})\}$

- Function *delltr* removes literal $sel(task(s)) \in \{s\}$ from each conjunctive clause of a context condition:

$$delltr : DNF \times S \rightarrow DNF$$

such that $\forall c^{\vee} = c_1^{\wedge} \vee c_2^{\wedge} ... \vee c_p^{\wedge} \in DNF,\ \forall s \in S,$

$$delltr(c^{\vee}, s) = c_1^{\wedge\prime} \vee c_2^{\wedge\prime} ... \vee c_p^{\wedge\prime} \in DNF$$

where $ltr(c_i^{\wedge\prime}) = ltr(c_i^{\wedge}) \setminus \{(sel(task(s)) \in \{s\})\}$

- Function *addcls* adds a particular conjunctive clause to each conjunctive clause of a context condition:

$$addcls : DNF \times DNF \rightarrow DNF$$

such that $\forall c^{\vee} = c_1^{\wedge} \vee c_2^{\wedge} ... \vee c_p^{\wedge} \in DNF,\ \forall c^{\wedge} \in DNF,$

$$addcls(c^{\vee}, c^{\wedge}) = c_1^{\wedge\prime} \vee c_2^{\wedge\prime} ... \vee c_p^{\wedge\prime} \in DNF \ \text{ where } \ ltr(c_i^{\wedge\prime}) = ltr(c_i^{\wedge}) \cup ltr(c^{\wedge})$$

- Function *delcls* removes a particular conjunctive clause from each conjunctive clause of a context condition:

$$delcls : DNF \times DNF \rightarrow DNF$$

such that $\forall c^{\vee} = c_1^{\wedge} \vee c_2^{\wedge} ... \vee c_p^{\wedge} \in DNF,\ \forall c^{\wedge} \in DNF,$

$$delcls(c^{\vee}, c^{\wedge}) = c_1^{\wedge\prime} \vee c_2^{\wedge\prime} ... \vee c_p^{\wedge\prime} \in DNF \ \text{ where } \ ltr(c_i^{\wedge\prime}) = ltr(c_i^{\wedge}) \setminus ltr(c^{\wedge})$$

Notation	Description
DNF	Conditions in disjunctive normal form
S^c	Correlation-aware service space
$addcls$	DNF condition after a clause addition to each comprising clause
$addltr$	DNF condition after a literal addition to each comprising clause
cls	Conjunctive clauses comprising a condition in DNF
$clst$	Conjunctive clauses constraining a task
$candidates^c$	Task's candidate services in S^c
ctx	Service's context in S^c
$delcls$	DNF condition after a clause deletion from each comprising clause
$delltr$	DNF condition after a literal deletion from each comprising clause
$lrange$	Literal's range
ltr	Literals comprising a conjunctive clause
$ltask$	Literal's task
$pltr$	Positive literals comprising a conjunctive clause
$qcomb$	Service's quality combinations
sel	Service selected for performing a task
$value_s^c$	Service's quality values in S^c
$task$	Task a service is performing
$cond$	Context condition associated with providing an attribute value by a service
$values$	Values offered by a service for an attribute

FIGURE 7.4: Additional notation used for the correlation-aware model

7.4 Correlation-aware Search Space Reduction

Pruning the candidate space per task is essential to reduce the computational cost of service selection. As was seen in Chapter 4, an effective way to achieve this is by using *request-based domination*, where service $s_1 \in candidates(t)$ is request-based dominated by service $s_2 \in candidates(t)$ iff s_2 is better for all *constrained quality attributes* and *utility*. In the absence of quality correlations among services, all request-based dominated services can simply be filtered out prior to selection, without affecting the ability to find an optimal solution. However, such domination-based elimination becomes more complex when quality correlations exist between services. This is because the quality values offered by a service may vary according to the composition context (i.e. the services selected for other tasks), and hence $s_1 \in candidates(t)$ might be request-based dominated by $s_2 \in candidates(t)$ in some composition contexts, but incomparable with (or even dominating) s_2 in other contexts. Consequently, in order to accurately assess domination among services, all their possible quality value combinations corresponding

to alternative composition contexts should be identified first. These quality combinations are introduced next, followed by the correlation-aware domination pruning of services.

7.4.1 Quality Combinations of Services

The set of possible quality value combinations, offered by service $s \in S$, for a number of different quality attributes $(a_1, ..., a_m) \in name_s(s)^m$, can be given as follows:

$$qcomb(s, (a_1, ..., a_m)) = \{((v_1, ..., v_m), c^{\vee}) \mid (v_1, ..., v_m) \in values(s, a_1) \times ... \times values(s, a_m) \ \wedge \ c^{\vee} = cond(s, a_1, v_1) \cap^{dnf} ... \cap^{dnf} cond(s, a_m, v_m) \neq \emptyset\}$$

where each $c^{\vee}$ represents the least constrained context condition in DNF for the provisioning of the respective quality values $(v_1, ..., v_m)$ by service s (see Figure 7.5 for definitions of the operators used). For example, in the scenario of Section 7.2, the least constrained context condition associated with providing values (pr:50, rel:97) by service *trsmC* is: $c^{\vee} = sel(\text{Book hotel}) \in \{\text{htlA}\}$. These context conditions $\{c^{\vee}\}$ are pairwise disjoint, i.e. at most one quality combination of service s is applicable in any service composition (the proof is straightforward from the definition of operator $\cap^{dnf}$, and therefore omitted). Thus, the number of possible composite services for the requested task $|comp(task_r)|$, is not affected by the number of quality combinations per service. Figure 7.6 shows an example of possible price and execution time combinations of a service s. Note that combination (pr:20, ex:20) is not possible since $cond(s, \text{pr}, 20) \cap^{dnf} cond(s, \text{ex}, 20) = \emptyset$.

Each possible quality combination $((v_1, ..., v_m), c^{\vee})$ of service s can be seen as a separate candidate service, providing the same functionality as s, with quality values

$(v_1, ..., v_m)$, and only applicable in context $c^{\vee}$. In other words, a new *correlation-aware candidate space* S^c, can be derived by replacing each service in S with its possible quality combinations for the attributes of interest. The quality values, and context functions associated with services in this space, are denoted $value_s^c$, and ctx, respectively; candidate services for task t are given by function $candidates^c(t) \subset S^c$. Given such candidate space transformation, the services in the context conditions should also be replaced with representatives (quality combinations) from S^c.

7.4.2 Correlation-aware Domination Pruning

Based on the new candidate space S^c, where each service is associated with one possible quality value combination, and an applicability context, domination-based pruning of services can be realised through restricting their applicability context, as follows. When service $s_1 \in candidates^c(t)$ is request-based dominated by service $s_2 \in candidates^c(t)$, the context of the former, $ctx(s_1)$, should be further restricted by eliminating from it all service compositions in which s_2 can replace s_1 (such compositions are guaranteed to be dominated, and thus are not worth considering). For example, suppose $ctx(s_1) = \text{ALL}$, and $ctx(s_2) = \neg(sel(t') \in \{s_3\})$. Being dominated by s_2, s_1 should be modified so that its context becomes $ctx(s_1) = (sel(t') \in \{s_3\})$. In this way, all service compositions containing s_1, but not containing s_3, are eliminated from the problem search space without affecting the ability to find an optimal solution because, for each of these compositions, a better composition can be derived by replacing s_1 with s_2. Notice that all service compositions including both s_1 and s_3 are still potential candidates for the optimal solution, since s_2 cannot replace s_1 in such compositions. Now, if s_1 was the dominating service instead (i.e. s_1 request-based dominates s_2), s_2 should be removed entirely from the candidate space of task t due to its context being included in $ctx(s_1)$.

Binary Operator $\cap^{l}$: associative, commutative, and idempotent

returns the greatest overlap of two literals concerning the same task (i.e. the least constrained literal satisfying both operands):

$$
\begin{array}{llll}
(sel(t) \in S_i) & \cap^{l} & (sel(t) \in S_j) & = \ (sel(t) \in S_i \cap S_j) \\
(sel(t) \in S_i) & \cap^{l} & \neg(sel(t) \in S_j) & = \ (sel(t) \in S_i \setminus S_j) \\
\neg(sel(t) \in S_i) & \cap^{l} & (sel(t) \in S_j) & = \ (sel(t) \in S_j \setminus S_i) \\
\neg(sel(t) \in S_i) & \cap^{l} & \neg(sel(t) \in S_j) & = \ \neg(sel(t) \in S_i \cup S_j)
\end{array}
$$

Operator Result: either a literal, or value $\emptyset$ in case the operands are **disjoint**:
$l_i \cap^{l} l_j = \emptyset \ \Leftrightarrow \ lrange(l_i \cap^{l} l_j) = \emptyset$

Binary Operator $\cap^{cls}$: associative, commutative, and idempotent

returns the greatest overlap of two conjunctive clauses, $c_i^{\wedge}$ and $c_j^{\wedge}$ (i.e. the least constrained conjunctive clause satisfying both operands):

$$
\begin{array}{ll}
ltr(c_i^{\wedge} \cap^{cls} c_j^{\wedge}) = & \{l_i \in ltr(c_i^{\wedge}) \mid \forall l_j \in ltr(c_j^{\wedge}), \ \ ltask(l_i) \neq ltask(l_j)\} \ \cup \\
& \{l_j \in ltr(c_j^{\wedge}) \mid \forall l_i \in ltr(c_i^{\wedge}), \ \ ltask(l_j) \neq ltask(l_i)\} \ \cup \\
& \{l_i \cap^{l} l_j \mid (l_i \in ltr(c_i^{\wedge})) \wedge (l_j \in ltr(c_j^{\wedge})) \wedge (ltask(l_i) = ltask(l_j))\}
\end{array}
$$

Operator Result: either a conjunctive clause, or value $\emptyset$ in case the operands are **disjoint**:
$c_i^{\wedge} \cap^{cls} c_j^{\wedge} = \emptyset \ \Leftrightarrow$
- the operands contain contradicting literals, i.e.

$$\exists l_i \in ltr(c_i^{\wedge}), \ \exists l_j \in ltr(c_j^{\wedge}), \ \ (ltask(l_i) = ltask(l_j)) \wedge (l_i \cap^{l} l_j = \emptyset)$$

- **or** no plan can contain all the tasks to satisfy both operands, i.e.

$$\forall p \in plan(root_t), \ \exists l \in pltr(c_i^{\wedge} \cap^{cls} c_j^{\wedge}), \ \ ltask(l) \notin nodes(p)$$

Binary Operator $\cap^{dnf}$: associative, commutative, and idempotent

returns the greatest overlap of two conditions in DNF, $c_i^{\vee}$ and $c_j^{\vee}$ (i.e. the least constrained condition in DNF satisfying both operands):

$$
\begin{array}{l}
cls(c_i^{\vee} \cap^{dnf} c_j^{\vee}) = \{c^{\wedge} = c_i^{\wedge} \cap^{cls} c_j^{\wedge} \mid (c_i^{\wedge} \in cls(c_i^{\vee})) \ \wedge \ (c_j^{\wedge} \in cls(c_j^{\vee})) \ \wedge \ (c^{\wedge} \neq \emptyset) \ \wedge \\
\qquad\qquad (\forall c_k^{\wedge} \in cls(c_i^{\vee} \cap^{dnf} c_j^{\vee}) \setminus \{c^{\wedge}\}, \ \ c_k^{\wedge} \cap^{cls} c^{\wedge} \neq c^{\wedge})\}
\end{array}
$$

(the last condition verifies that $c^{\wedge}$ is not a specific version of $c_k^{\wedge}$)

Operator Result: either a condition in DNF, or value $\emptyset$ in case the operands are **disjoint**:
$c_i^{\vee} \cap^{dnf} c_j^{\vee} = \emptyset \ \Leftrightarrow \ cls(c_i^{\vee} \cap^{dnf} c_j^{\vee}) = \emptyset$

Note: $\forall c^{\vee}$, $\quad c^{\vee} \cap^{dnf} \text{ALL} = c^{\vee}$
$\qquad\qquad\qquad\quad\ \ c^{\vee} \cap^{dnf} \emptyset = \emptyset$

Unary Operator $\neg^{dnf}$:

returns the negation of a condition in DNF, $c^{\vee} = c_1^{\wedge} \wedge c_2^{\wedge} \wedge ... \wedge c_p^{\wedge}$:

$$
\begin{array}{l}
cls(\neg^{dnf}(c^{\vee})) = \{c_{ng}^{\wedge} \mid \exists l_1 \in ltr(c_1^{\wedge}), \ \exists l_2 \in ltr(c_2^{\wedge}) \ , ... \ , \ \exists l_p \in ltr(c_p^{\wedge}), \\
\qquad\qquad\qquad\qquad\qquad\qquad c_{ng}^{\wedge} = \neg l_1 \cap^{cls} \neg l_2 \cap^{cls} \ ... \ \cap^{cls} \neg l_p \neq \emptyset\}
\end{array}
$$

Operator Result: either a condition in DNF, or value $\emptyset$: $\neg^{dnf}(c^{\vee}) = \emptyset \ \Leftrightarrow \ cls(\neg^{dnf}(c^{\vee})) = \emptyset$

Note: $\quad \neg^{dnf}(\text{ALL}) = \emptyset$
$\qquad\quad\ \ \neg^{dnf}(\emptyset) = \text{ALL}$

FIGURE 7.5: The definition of operators $\cap^{l}$, $\cap^{cls}$, $\cap^{dnf}$, and $\neg^{dnf}$

$$value_s(s, \text{pr}) = \begin{cases} 10 & \text{if } (sel(t_1) \in \{s_1\}) \\ 20 & \text{if } \neg(sel(t_1) \in \{s_1\}) \end{cases}$$

$$value_s(s, \text{ex}) = \begin{cases} 20 & \text{if } (sel(t_1) \in \{s_1\}) \wedge (sel(t_2) \in \{s_2\}) \\ 30 & \text{if } \neg(sel(t_1) \in \{s_1\}) \vee \neg(sel(t_2) \in \{s_2\}) \end{cases}$$

$$qcomb(s) = \begin{cases} (10, 20) & \text{if } (sel(t_1) \in \{s_1\}) \wedge (sel(t_2) \in \{s_2\}) \\ (10, 30) & \text{if } (sel(t_1) \in \{s_1\}) \wedge \neg(sel(t_2) \in \{s_2\}) \\ (20, 30) & \text{if } \neg(sel(t_1) \in \{s_1\}) \end{cases}$$

FIGURE 7.6: Possible price and execution time combinations of service s

Applying such domination-based context restriction based only on services' *individual* contexts, without considering additional constraints that could be imposed on a service's context by other services, might result in eliminating interesting (non-dominated) compositions. Consider, for instance, the scenario of Section 7.2, where the candidate space S^c can be derived by replacing service *trsmC* with two services: service $trsmC_1$(pr:90, rel:97) with context $\neg(sel(\text{Book hotel}) \in \{\text{htlA}\})$, and service $trsmC_2$(pr:50, rel:97) with context $sel(\text{Book hotel}) \in \{\text{htlA}\}$, and assigning an unrestricted (i.e. ALL) context to all other services. Here, ignoring the fact that *htlB* cannot be used with $trsmC_2$ (a constraint imposed by *trsmC*), and keeping its context unrestricted, will lead to filtering out service *htlA* (since *htlA* is dominated by *htlB*, and their assumed contexts are the same), thus eliminating the optimal solution '*arlnA*, *htlA*, *txi*, $trsmC_2$' from the search space. It is thus important to modify *htlB*'s context to $\neg(sel(\text{book sightseeing}) \in \{\text{trsm}C_2\})$, so that, when domination pruning is performed, the context of the dominated *htlA* is restricted to $sel(\text{book sightseeing}) \in \{\text{trsm}C_2\}$, instead of removing *htlA* entirely from the candidate space.

Formally, given service $s \in candidates^c(t)$, the services possibly affecting its context, denoted $affctx(s)$, are those referring to task t in their applicability contexts, i.e.

$$(affctx(s) \subset S^c \setminus candidates^c(t)) \wedge (\forall s' \in affctx(s), \quad clst(ctx(s'), t) \neq \emptyset)$$

The constraints of each service $s' \in affctx(s)$, are reflected in $ctx(s)$, according to the following cases (from the perspective of s').

Case 1. $\forall c^{\wedge} \in clst(ctx(s'), t), \quad c^{\wedge} \cap^{cls} (sel(t) \in \{s\}) \neq \emptyset$, indicating that it is always possible to instantiate task t with service s, in any composition including task t and containing service s' (satisfying the context condition of s'). Hence, no additional constraints need be added to $ctx(s)$.

Case 2. $\exists c^{\wedge} \in clst(ctx(s'), t), \quad c^{\wedge} \cap^{cls} (sel(t) \in \{s\}) = \emptyset$, indicating that, of all compositions containing service s' (satisfying the context condition of s'), and performing task t, there exist some compositions where it is not possible to instantiate t with service s. Hence, to differentiate s from t's other candidates that can be used in such composition contexts of s', $ctx(s)$ should be updated to:

$$ctx_d(s) = delltr(addltr(ctx(s), s) \cap^{dnf} (\neg(sel(task(s')) \in \{s'\}) \vee ctx(s')), s) \qquad (7.1)$$

where ctx_d denotes the context for domination purposes. Note that, in order to keep the context conditions concise, the goal in these cases is to capture the necessary differences in t's services applicability that will prevent elimination of interesting compositions during pruning, rather than a full restriction of their contexts according to other services.

The quality combinations of services and their adjusted contexts ctx_d, can be precomputed (before request time), and continuously modified in response to changes in services (addition of new services, deletion of existing services, or changes in the quality values of services).

Now, given two services $s_1, s_2 \in candidates^c(t)$ such that s_1 is request-based dominated by s_2, the context of s_1 should be restricted so that all the service compositions containing service s_1, and satisfying context $ctx_d(s_1) \cap^{dnf} ctx_d(s_2)$, are removed from

the search space (due to being dominated):

$$ctx_d(s_1) := \quad ctx_d(s_1) \setminus^{dnf} ctx_d(s_2) \;=\; ctx_d(s_1) \cap^{dnf} (\neg^{dnf}(ctx_d(s_2)))$$

(see Figure 7.5 for the definition of operator $\neg^{dnf}$).

Notice that if $ctx_d(s_1) \setminus^{dnf} ctx_d(s_2) = \emptyset$, service s_1 should be eliminated entirely from task t's candidates. The candidate services of task $t \in T$ surviving the pruning, are denoted $rcandidates^c(t)$, where each service $s \in rcandidates^c(t)$ is non-dominated in its associated context $ctx_d(s)$ (i.e. $ctx_d(s)$ is pruned until no other candidate service of t is both applicable in $ctx_d(s)$, and request-based dominates service s).

7.4.3 Inter-task Pruning

Although domination-based pruning removes from the context of each service $s \in candidates^c(t)$, the compositions guaranteed to be dominated from task t's perspective, this context might still refer to uninteresting compositions when considering other tasks. Such uninteresting compositions are those not possible, either due to some of their component services (candidates of other tasks) being eliminated during domination pruning, or due to the conditions imposed on these components' context (originally by the service provider, or as a result of domination-based context restriction). For example, consider the abstract plan of Figure 7.1, and assume the candidate services $candidates^c$, of tasks C and E, are as provided in Figure 7.7, with their context conditions before and after domination pruning being ctx_d^{bf} and ctx_d^{af}, respectively (for simplicity, only the range is shown for each literal). As can be seen, $ctx_d^{af}(s_{C1})$ is not satisfiable since, according to $ctx_d^{af}(s_{E1})$, s_{E1} cannot be used simultaneously with both s_{C1} and s_{D1}. Thus, further pruning can be achieved by eliminating s_{C1} entirely from

Task C	ctx	ctx_d^{bf}	ctx_d^{af}
s_{C1}	s_{D1}	s_{D1}	$s_{D1} \wedge s_{E1}$
s_{C2}	$\neg s_{D1}$	$\neg s_{D1}$	$\neg s_{D1}$
s_{C3}	s_{E1}	s_{E1}	$\emptyset$
s_{C4}	$\neg s_{E1}$	$\neg s_{E1}$	$\neg s_{E1}$
s_{C5}	ALL	ALL	ALL
Task E	ctx	ctx_d^{bf}	ctx_d^{af}
s_{E1}	ALL	$\neg s_{C4}$	$(\neg s_{C4} \wedge \neg s_{D1}) \vee s_{C3}$
s_{E2}	s_{D1}	$s_{D1} \wedge \neg s_{C3}$	$s_{D1} \wedge \neg s_{C3}$
s_{E3}	$\neg s_{D1}$	$\neg s_{D1} \wedge \neg s_{C3}$	$\neg s_{D1} \wedge \neg s_{C3}$

FIGURE 7.7: Context conditions of services before and after domination pruning, assuming s_{C1} is dominated by s_{C4}, s_{C3} is dominated by s_{C5}, and s_{E1} is dominated by s_{E2}

C's candidates (this does not affect solution optimality because any composition containing services s_{C1}, s_{D1} and s_{E1}, is always dominated by a composition replacing s_{C1} and s_{E1} with s_{C4} and s_{E2}, respectively). Additionally, $ctx_d^{af}(s_{E1})$ should be further restricted to $\neg(sel(C) \in \{s_{C4}\}) \wedge \neg(sel(D) \in \{s_{D1}\})$, since s_{C3} is no longer considered task C's candidate ($ctx_d^{af}(s_{C3}) = \emptyset$). This inter-task based deletion of infeasible compositions from service contexts avoids unnecessary computations during selection, especially in selection approaches where the composite solution is built incrementally (e.g., if s_{C1} remains a candidate for task C in graph-based approaches, then all possible instantiations of sub-plans ACD and BCD containing this service will be computed, before concluding that these sub-compositions cannot be combined with any candidate of task E, and therefore are invalid).

The context of each service $s \in rcandidates^c(t)$, can be checked for infeasible compositions, as follows. Given clause $c^\wedge \in cls(ctx_d(s))$, literal $l \in ltr(c^\wedge)$, and service $s_{de} \in lrange(l)$, s_{de} should be eliminated from $lrange(l)$, i.e. $lrange(l) := lrange(l) \setminus \{s_{de}\}$, if: s_{de} is pruned out from task $ltask(l)$'s candidates, i.e. $s_{de} \notin rcandidates^c(ltask(l))$; or l is a positive propositional variable, and $ctx_d(s_{de})$ is not consistent with $c^\wedge$, i.e. $l \in pltr(c^\wedge) \wedge (ctx_d(s_{de}) \cap^{dnf} addltr(c^\wedge, s) = \emptyset)$, indicating that $c^\wedge$ cannot be satisfied

when s_{de} instantiates task $ltask(l)$. Now, if $lrange(l)$ is empty as a result of s_{de} elimination, two cases are distinguished. Case 1: literal l is positive, implying that $c^{\wedge}$ is no longer satisfiable, and should be removed from $cls(ctx_d(s))$, and if the latter consequently becomes empty, service s is not applicable in any context, and should be removed from $rcandidates^c(t)$. Case 2: literal l is negative, implying that l should simply be removed from $ltr(c^{\wedge})$, and if the latter consequently becomes empty, $ctx_d(s)$ should be updated to ALL since, in this case, $c^{\wedge}$ is always satisfiable.

Such inter-task pruning of service context conditions after domination pruning is specified in Algorithm 9, where: $dependents(s) \subset S^c$ are the services dependent on service s, i.e. those referring to s in their context conditions (the dependants are assigned to each service when calculating the quality combinations, and updated during domination-based context restriction); and procedure $adjust - dependents(s, type)$ recursively adjusts the context conditions of service s's dependants, in response to the elimination (type='rmv') and further context restriction (type='ch') of s.

Algorithm 9 inter-task-pruning

```
1: for each t ∈ T do
2:   for each s ∈ rcandidates^c(t) do
3:     tag s as processed
4:     remove infeasible compositions from ctx_d(s)
5:     if ctx_d(s) = ∅ then
6:       rcandidates^c(t) ← rcandidates^c(t) \ {s}
7:       adjust-dependents(s, 'rmv')
8:     else if ctx_d(s) is restricted then
9:       adjust-dependents(s,'ch')
```

7.5 Correlation-aware Selection Algorithm

Given set $rcandidates^c(v)$ for each task node v, the correlation-aware extension of our selection algorithm in Chapter 4 can be summarised as follows. Each node v in the plan paths graph stores the optimal instances, denoted as $oinstances(v, p_v)$, for each

Procedure 10 adjust-dependents(s_{de}, type)

1: **for** each processed and non eliminated $s \in dependents(s_{de})$ **do**
2: **if** type='rmv' **then**
3: remove s_{de} from $ctx_d(s)$
4: **else**
5: remove s_{de} from $ctx_d(s)$ when inconsistent
6: **if** $ctx_d(s) = \emptyset$ **then**
7: $rcandidates^c(task(s)) \leftarrow rcandidates^c(task(s)) \setminus \{s\}$
8: adjust-dependents(s,'rmv')
9: **else if** $ctx_d(s)$ is restricted **then**
10: adjust-dependents(s,'ch')

path $p_v + v$ discoved so far from the start node to v, such that $p_v \in validpred(v)$. Similarly to atomic services, each instance ins (a combination of one or more services), is associated with an applicability context $ctx(ins)$ (defined in terms of the context of its component services), and is considered optimal (non-dominated) if no other possible instance of the same path has both better values for all the constrained attributes and better utility while being applicable in $ctx(ins)$. The graph nodes are traversed in topological order, and when visiting edge (v, u), the optimal instances stored at v are combined with u's candidate services, as follows: $\forall ins \in oinstances(v, p_v)$ s.t. $p_v + v \in validpred(u)$, $\forall s \in rcandidates^c(u)$, ins is composed with s iff their context conditions are not disjoint, i.e. $c^{\vee}_{ins+s} = addcls(ctx(ins), ins) \cap^{dnf} addltr(ctx_d(s), s) \neq \emptyset$, where ins is considered a conjunction of positive propositional variables whose ranges correspond to ins's component services. The resulting instance $ins + s$ is assigned context $ctx(ins + s) = delcls(c^{\vee}_{ins+s}, ins + s)$, and if optimal, is stored at node u. This algorithm is realised by replacing the *process-edge* procedure (Procedure 2) of Algorithm 1 with Procedure 11.

Procedure 11 crl-process-edge(v,u)

for each $p_u \in validpred(u)$ **do**
  if $enode(p_u) = v$ **then**
    for each $s \in rcandidates^c(u)$ **do**
      for each optimal instance $ins_v \in oinstances(v, p_u - v)$ **do**
        if instance $ins_v + s$ is satisfactory **then**
          $c^{\vee} \leftarrow addcls(ctx(ins_v), ins_v) \cap^{dnf} addltr(ctx_d(s), s)$
          if $c^{\vee} \neq \emptyset$ **then**
            $ctx(ins_v + s) \leftarrow delcls(c^{\vee}, ins_v + s)$
            crl-check-instance-optimality($ins_v + s$,u,p_u)

Procedure 12 crl-check-instance-optimality(ins,u,p_u)

optml $\leftarrow 1$
for each optimal instance $ins_u \in oinstances(u, p_u)$ **do**
  if ins_u r-dm ins **then**
    $ctx(ins) \leftarrow ctx(ins) \backslash^{dnf} ctx(ins_u)$
    if $ctx(ins) = \emptyset$ **then**
      optml $\leftarrow 0$
      break
  else if ins r-dm ins_u **then**
    $ctx(ins_u) \leftarrow ctx(ins_u) \backslash^{dnf} ctx(ins)$
    if $ctx(ins_u) = \emptyset$ **then**
      $oinstances(u, p_u) \leftarrow oinstances(u, p_u) \setminus \{ins_u\}$
if optml=1 **then**
  $oinstances(u, p_u) \leftarrow oinstances(u, p_u) \cup \{ins\}$

7.6 Analytical Study

This section studies the time complexity of the proposed correlation-aware selection algorithm. For simplicity, the analysis is first provided for the specific case of one abstract plan, and later generalised to handle the case of the planning knowledge hierarchy (i.e. multiple abstract plans). The notation used throughout the analysis is summarised in Figure 7.8.

7.6.1 One Abstract Plan

Consider a sequential abstract plan comprised of k tasks, $v_1 v_2 ... v_k$, where each has nc available candidate services in the correlation-aware candidate space S^c. The time

Notation	Description
n	the number of candidates per task in the original candidate space S
k	the number of tasks per abstract plan
$crlpcg$	the correlation percentage per dependent task in S, i.e. the percentage of services with dependent attributes per task
X	the number of quality combinations per dependent service in S
$crltsk$	the number of dependent tasks (tasks with non-zero correlation percentage) excluding the current task
nc	the number of candidates per task in the correlation-aware candidate space S^c, i.e. $nc = n + n \times crlpcg \times (X - 1)$
C_0	the number of conjunctive clauses in the original context condition, $ctx(s)$, of a dependent service $s \in S^c$
L_0	the number of literals per conjunctive clause in the original context condition, $ctx(s)$, of a dependent service $s \in S^c$
C	the number of conjunctive clauses in a context condition
PC	the number of positive conjunctive clauses (clauses with at least one positive literal) in a context condition
L	the number of literals in a conjunctive clause
PL	the number of positive literals in a conjunctive clause
NL	the number of negative literals in a conjunctive clause
R	the number of services in a literal's range
$\|c^\vee\|$	the size (number of conjunctive clauses) of context condition $c^\vee$

FIGURE 7.8: Notation used throughout the analysis

complexity of our correlation-aware selection algorithm, $\tau(alg_{crl})$, can be computed in terms of the domination pruning time, $\tau(dprn)$, inter-task pruning time, $\tau(iprn)$, and selection time, $\tau(sel)$, as follows:

$$\tau(alg_{crl}) = \tau(dprn) + \tau(iprn) + \tau(sel)$$

To prune the dominated services for task node v_i, each candidate of this task may need to be compared with all the remaining candidates, i.e. there are $nc \times (nc - 1)$ service comparisons in the worst case. Hence, $\tau(dprn)$ is $O(nc^2 \times \tau(ctxmns))$, where $\tau(ctxmns)$ represents the time complexity of computing the difference ($\setminus^{dnf}$) for two context conditions (see Figure 7.9 for the time complexity of context-related operations).

In inter-task pruning, the context of each service $s \in rcandidates^c(v_i)$, $ctx_d(s)$, is

Operation	**Time Complexity**
Literal Intersection $\cap^{l}$, $\tau(ltrintr)$	$O(R^2)$
Clause Intersection $\cap^{cls}$, $\tau(clsintr)$	$O(L^2 + L \times R^2)$
Condition Intersection $\cap^{dnf}$, $\tau(ctxintr)$	$O(C^2 \times (L^2 + L \times R^2))$
Condition Negation $\neg^{dnf}$, $\tau(ctxneg)$	$O(L^C \times (C-1) \times (R^2 + min(C-1, k)))$
Condition Difference $\setminus^{dnf}$, $\tau(ctxmns)$	$\tau(ctxintr) + \tau(ctxneg)$
Dependee Consistency Check $\tau(ctxcns)$	$O(C \times (L^2 + L \times R^2))$

FIGURE 7.9: Time complexity of context-related operations

validated by removing from this context the dependee services eliminated during domination pruning, and checking those not eliminated and positive (i.e. belonging to positive literals) for consistency. The consistency check of a positive dependee service s_{de} in $ctx_d(s)$ may need to be repeated each time the context of service s_{de}, $ctx_d(s_{de})$, is restricted, depending on whether the restriction occurs after the processing of service s (such a situation is encountered when the dependee services appear after the dependants in the processing order). Now, since there could be at most $\beta = PC \times (1 + PL \times (R-1))$ consecutive context restrictions for a service during inter-task pruning (each corresponding to the elimination of a positive dependee service from the service's context condition), each positive dependee service s_{de} in $ctx_d(s)$ is checked for consistency β times in the worst case. Hence:

$$\tau(iprn) \text{ is } O(k \times nc \times [C \times (NL \times R + PL \times R \times \beta \times \tau(ctxcns))])$$

where $\tau(ctxcns)$ is the time complexity of performing a consistency check (see Figure 7.9).

To select the optimal solution (the best instance of path $v_1 v_2 ... v_k$), each node $v_{i>1}$ records the optimal instances of path $v_1 v_2 ... v_i$, denoted $oinstances(v_i)$. Hence:

$$\tau(sel) = \sum_{i=2}^{k} \tau(oinstances(v_i)) \tag{7.2}$$

The time required to calculate $oinstances(v_i)$, $\tau(oinstances(v_i))$, depends on the sizes

of $oinstances(v_{i-1})$ and $rcandidates^c(v_i)$, and on the context conditions associated with the corresponding instances and services. The former can be defined in terms of the following rates.

- The service pruning rate, $spr_i \in [0, 1]$, denotes the percentage of candidate services pruned *entirely* for node v_i prior to selection. That is, $|rcandidates^c(v_i)| = nc \times sr_i$, where $sr_i = 1 - spr_i$.

- The instance validity rate, $vr_i \in [0, 1]$, denotes the percentage of *valid* instances (service combinations) at node v_i (of $|oinstances(v_{i-1})| \times |rcandidates^c(v_i)|$ candidates). This is determined based on the context conditions of node v_i's services and node v_{i-1}'s optimal instances, which are either originally imposed by the service providers or assigned as a result of the formerly-performed pruning.

- The instance pruning rate, $ipr_i \in [0, 1]$, denotes the percentage of valid candidate instances pruned *entirely* for path $v_1 v_2 ... v_i$, when computing the optimal instances at node v_i. That is:

$$|oinstances(v_i)| = |oinstances(v_{i-1})| \times |rcandidates^c(v_i)| \times vr_i \times ir_i \qquad (7.3)$$

  (where $ir_i = 1 - ipr_i$)

  Since $|oinstances(v_1)| = |rcandidates^c(v_1)| = nc \times sr_1$, Equation 7.3 becomes as follows:

$$|oinstances(v_i)| = nc^i \times \prod_{m=1}^{i} sr_m \times \prod_{m=2}^{i} vr_m \times \prod_{m=2}^{i} ir_m \qquad (7.4)$$

Based on this, $\tau(oinstances(v_{i>1}))$, which involves identifying the valid instances at node v_i (Line 7.5), and checking their optimality (Line 7.6), can be computed as follows:

$$\text{is } O(|oinstances(v_{i-1})| \times |rcandidates^c(v_i)| \times \tau(ctxintr) \quad + \tag{7.5}$$

$$|oinstances(v_{i-1})| \times |rcandidates^c(v_i)| \times vr_i \times |oinstances(v_i)| \times \tau(ctxmns)) \tag{7.6}$$

where $\tau(ctxintr)$ and $\tau(ctxmns)$ represent the time complexity of computing the intersection ($\cap^{dnf}$) and difference ($\setminus^{dnf}$), respectively, for two context conditions.

Since $\tau(ctxmns) > \tau(ctxintr)$ (see Figure 7.9), and assuming $vr_i \times |oinstances(v_i)| \geq 1$ (usually holds for a sufficiently large number of candidates nc), the first term (Line 7.5) of $\tau(oinstances(v_i))$ is dominated by the second term (Line 7.6), and thus can be ignored. Consequently, $\tau(oinstances(v_i))$:

$$\text{is } O(|oinstances(v_{i-1})| \times |rcandidates^c(v_i)| \times vr_i \times |oinstances(v_i)| \times \tau(ctxmns))$$

$$\text{is } O(nc^{2i} \times \prod_{m=1}^{i} sr_m^2 \times \prod_{m=2}^{i} vr_m^2 \times \prod_{m=2}^{i-1} ir_m^2 \times ir_i \times \tau(ctxmns)) \tag{7.7}$$

Based on this, selection complexity $\tau(sel)$ can be defined by the following lemmas.

Lemma 7.1. *Suppose* $\forall i \in [1,k],\ sr_i = sr$, *and* $\forall i \in [2,k],\ (ir_i = ir) \wedge (vr_i = vr)$. *Suppose also that* $sr \times ir \times vr > \frac{1}{nc}$. *Then,* $\forall i \in [2,k-1],\ \tau(oinstances(v_i)) < \tau(oinstances(v_{i+1}))$.

Proof.

$$
\begin{aligned}
& sr \times ir \times vr > \frac{1}{nc} \\
\Rightarrow\ & sr^2 \times ir^2 \times vr^2 > \frac{1}{nc^2} \\
\Rightarrow\ & \tau(oinstances(v_i)) \times sr^2 \times ir^2 \times vr^2 > \tau(oinstances(v_i)) \times \frac{1}{nc^2} \\
\Rightarrow\ & \tau(oinstances(v_i)) \times sr^2 \times ir^2 \times vr^2 \times nc^2 > \tau(oinstances(v_i)) \\
\Rightarrow\ & nc^{2(i+1)} \times sr^{2(i+1)} \times vr^{2(i+1)-2} \times ir^{2(i+1)-3} \times \tau(ctxmns) > \tau(oinstances(v_i)) \\
\Rightarrow\ & \tau(oinstances(v_{i+1})) > \tau(oinstances(v_i))
\end{aligned}
$$

□

Lemma 7.2. *Suppose* $\forall i \in [1,k],\ sr_i = sr$, *and* $\forall i \in [2,k],\ (ir_i = ir) \wedge (vr_i = vr)$. *Suppose also that* $sr \times ir \times vr \leq \frac{1}{nc}$. *Then,* $\forall i \in [2,k-1],\ \tau(oinstances(v_i)) \geq \tau(oinstances(v_{i+1}))$.

Proof. The same as Lemma 7.1. □

Lemma 7.3. *Suppose* $\forall i \in [1,k],\ sr_i = sr$, *and* $\forall i \in [2,k],\ (ir_i = ir) \wedge (vr_i = vr)$. *Suppose also that* $sr \times ir \times vr > \frac{1}{nc}$. *Then, our algorithm achieves a selection complexity* $\tau(sel)$ *of* $O(n^\alpha)$, *such that:*

$$\alpha = 2k + \log_n[(1 + crlpcg \times (X-1))^{2k} \times sr^{2k} \times vr^{2k-2} \times ir^{2k-3} \times \tau(ctxmns)]$$

Proof. From Lemma 7.1 and Equation 7.2, $\tau(sel)$ is $O(\tau(oinstances(v_k)))$ (assuming k is a small number, i.e. $k << n$). That is, $\tau(sel)$ is $O(n^\alpha)$, with:

$$n^\alpha = nc^{2k} \times sr^{2k} \times vr^{2k-2} \times ir^{2k-3} \times \tau(ctxmns) \quad \text{(see Equation 7.7)}$$

Now, since $nc = n + n \times crlpcg \times (X-1) = n(1 + crlpcg \times (X-1))$, n^{α} becomes as follows:

$$\begin{aligned} n^{\alpha} &= n^{2k} \times (1 + crlpcg \times (X-1))^{2k} \times sr^{2k} \times vr^{2k-2} \times ir^{2k-3} \times \tau(ctxmns) \\ \Rightarrow n^{\alpha-2k} &= (1 + crlpcg \times (X-1))^{2k} \times sr^{2k} \times vr^{2k-2} \times ir^{2k-3} \times \tau(ctxmns) \\ \Rightarrow \alpha &= 2k + \log_n[(1 + crlpcg \times (X-1))^{2k} \times sr^{2k} \times vr^{2k-2} \times ir^{2k-3} \times \tau(ctxmns)] \end{aligned}$$

□

Lemma 7.4. *Suppose* $\forall i \in [1,k],\ sr_i = sr$, *and* $\forall i \in [2,k],\ (ir_i = ir) \wedge (vr_i = vr)$. *Suppose also that* $sr \times ir \times vr \leq \frac{1}{nc}$. *Then, our algorithm achieves a selection complexity* $\tau(sel)$ *of* $O(n^{\alpha})$, *such that:*

$$\alpha = 4 + \log_n[(1 + crlpcg \times (X-1))^4 \times sr^4 \times vr^2 \times ir \times \tau(ctxmns)]$$

Proof. From Lemma 7.2 and Equation 7.2, $\tau(sel)$ is $O(\tau(oinstances(v_2)))$ (assuming k is a small number, i.e. $k << n$). That is, $\tau(sel)$ is $O(n^{\alpha})$, with:

$$n^{\alpha} = nc^4 \times sr^4 \times vr^2 \times ir \times \tau(ctxmns) \quad \text{(see Equation 7.7)}$$

Now, since $nc = n + n \times crlpcg \times (X-1) = n(1 + crlpcg \times (X-1))$, n^{α} becomes as follows:

$$\begin{aligned} n^{\alpha} &= n^4 \times (1 + crlpcg \times (X-1))^4 \times sr^4 \times vr^2 \times ir \times \tau(ctxmns) \\ \Rightarrow n^{\alpha-4} &= (1 + crlpcg \times (X-1))^4 \times sr^4 \times vr^2 \times ir \times \tau(ctxmns) \\ \Rightarrow \alpha &= 4 + \log_n[(1 + crlpcg \times (X-1))^4 \times sr^4 \times vr^2 \times ir \times \tau(ctxmns)] \end{aligned}$$

□

Example. Suppose that $n = 200$, $sr = vr = ir = 10\%$, $X = 2$, $crlpcg = 10\%$, and $\tau(ctxmns)$ is O(1). From Lemma 7.4, the selection process is of time complexity $O(n)$ (i.e. linear complexity), since $\alpha = 4 + \log_{200}((1.1)^4 \times (0.1)^7) \simeq 1$.

7.6.2 Multiple Abstract Plans

Given a planning knowledge hierarchy with l levels, d decomposition graphs per parent task, and k sub-tasks per decomposition graph, the number of alternative abstract plans for the goal task (root), $plnnum(l, d, k)$, is provided in Equation 4.6. Based on this, Lemmas 7.3 and 7.4 can be generalised as follows.

Lemma 7.5. *Suppose* $\forall i,\ sr_i = sr,\ ir_i = ir,$ *and* $vr_i = vr$. *Suppose also that* $sr \times ir \times vr > \frac{1}{nc}$. *Then, our algorithm achieves a selection complexity* $\tau(sel)$ *of* $O(n^\alpha)$, *such that:*

$$\alpha = 2 \times tmx + \log_n[pr \times plnnum(l, d, k) \times (1 + crlpcg \times (X - 1))^{2 \times tmx} \times sr^{2 \times tmx} \times vr^{2 \times tmx - 2} \times ir^{2 \times tmx - 3} \times \tau(ctxmns)]$$

where pr *is the percentage of abstract plans surviving plan-based pruning; and* $tmx \leq k^{l-1}$ *is the maximum number of tasks per abstract plan (among the surviving plans).*

Proof. The proof can be derived similarly to Lemma 7.3, given that $\tau(sel)$ is $O(pr \times plnnum(l, d, k) \times \tau(oinstances(v_{tmx})))$. □

Lemma 7.6. *Suppose* $\forall i,\ sr_i = sr,\ ir_i = ir,$ *and* $vr_i = vr$. *Suppose also that* $sr \times ir \times vr \leq \frac{1}{nc}$. *Then, our algorithm achieves a selection complexity* $\tau(sel)$ *of* $O(n^\alpha)$, *such that:*

$$\alpha = 4 + \log_n[pr \times plnnum(l, d, k) \times (1 + crlpcg \times (X - 1))^4 \times sr^4 \times vr^2 \times ir \times \tau(ctxmns)]$$

where pr is the percentage of abstract plans surviving plan-based pruning.

Proof. The proof can be derived similarly to Lemma 7.4, given that $\tau(sel)$ is $O(pr \times plnnum(l,d,k) \times \tau(oinstances(v_2)))$. □

7.6.3 Context Complexity

The aim of this section is to analyse how the pre-selection pruning steps increase the complexity of service context conditions, and to provide an upper-bound for such an increase. Since the number of literals per conjunctive clause is bounded by the number of tasks per plan, i.e. $L \leq k-1$, we focus next on analyzing the growth in the number of comprising conjunctive clauses C (referred to as the context condition *size*).

Lemma 7.7. *Given a service $s \in candidates^c(t)$, the pre-domination context restriction affects the size of this service's original context condition $ctx(s)$, by a factor of λ, i.e. $|ctx_d(s)| = |ctx(s)| \times \lambda$, such that:*

$$\lambda \leq (X \times (C_0 - 1) + 1)^{crlpcg \times n \times crltsk}$$

Proof. Consider a dependent service $s' \in candidates(t')$. In the worst case where all the quality combinations of service s', $s'_1, s'_2, ..., s'_X \in candidates^c(t')$, are considered affecting for service $s \in candidates^c(t)$ (i.e. they satisfy Case 2 of the pre-domination context adjustment), $ctx(s)$ should be replaced with $delltr(addltr(ctx(s), s) \cap^{dnf} ctx', s)$

(see Equation 7.1), where:

$$\begin{aligned} ctx' = & [\neg(sel(t') \in \{s'_1\}) \vee ctx(s'_1)] && \cap^{dnf} \\ & [\neg(sel(t') \in \{s'_2\}) \vee ctx(s'_2)] && \cap^{dnf} \\ & ... && \cap^{dnf} \\ & [\neg(sel(t') \in \{s'_X\}) \vee ctx(s'_X)] \end{aligned}$$

Since $\{ctx(s'_i)\}_i$ are pairwise disjoint, ctx' can be expressed as follows:

$$\begin{aligned} ctx' = & [\neg(sel(t') \in \{s'_i\}_i)] && \vee \\ & [\neg(sel(t') \in \{s'_i\}_{i\neq 1}) \cap^{dnf} ctx(s'_1)] && \vee \\ & ... && \vee \\ & [\neg(sel(t') \in \{s'_i\}_{i\neq X}) \cap^{dnf} ctx(s'_X)] \end{aligned}$$

From the definition of $\cap^{dnf}$, we have that:

$$|delltr(addltr(ctx(s), s) \cap^{dnf} ctx', s)| \ \leq \ |ctx(s)| \times |ctx'| \ \leq \ |ctx(s)| \times (1 + X \times C_0)$$

Since we assumed that for each s'_i, there exists $c^{\wedge} \in ctx(s'_i)$ such that $c^{\wedge} \cap^{cls} (sel(t) \in \{s\}) = \emptyset$, at least one clause can be eliminated from each $ctx(s'_i)$ in ctx' without affecting the result of $addltr(ctx(s), s) \cap^{dnf} ctx'$, leading to:

$$|delltr(addltr(ctx(s), s) \cap^{dnf} ctx', s)| \ \leq \ |ctx(s)| \times (1 + X \times (C_0 - 1))$$

In the worst case scenario, the context conditions of all dependent services ($n \times crlpcg$) for all dependent tasks excluding task t ($crltsk$) should be reflected similarly on $ctx(s)$, resulting in $|ctx_d(s)| \leq |ctx(s)| \times (1 + X \times (C_0 - 1))^{n \times crlpcg \times crltsk}$. □

Lemma 7.8. *Given a service $s \in candidates^c(t)$, the domination-based context restriction affects the size of this service's pre-domination context condition $ctx_d^{bf}(s)$, by a factor of ω, i.e. $|ctx_d^{af}(s)| = |ctx_d^{bf}(s)| \times \omega$, such that:*

$$\omega \leq ({L_0}^{C_0} + (k-1)^{\lambda})^{nc-1}$$

Proof. When service $s \in candidates^c(t)$ is dominated by service $s' \in candidates^c(t)$, the context of service s is restricted as follows:

$$ctx_d^{af}(s) = ctx_d^{bf}(s) \cap^{dnf} (\neg^{dnf}(ctx_d^{bf}(s'))) \tag{7.8}$$

According to the definition of operators $\cap^{dnf}$ and $\neg^{dnf}$, the following holds:

$$\begin{aligned} |\neg^{dnf}(c_i^{\vee} \cap^{dnf} c_j^{\vee})| &\leq |\neg^{dnf}(c_i^{\vee}) \vee \neg^{dnf}(c_j^{\vee})| \\ &\leq |\neg^{dnf}(c_i^{\vee})| + |\neg^{dnf}(c_j^{\vee})| \end{aligned}$$

Based on this, and since $ctx_d^{bf}(s') = ctx(s') \cap^{dnf} ctx^{dt}$, where ctx^{dt} corresponds to the context reflected on service s' by the pre-domination context adjustment step, we have the following:

$$|\neg^{dnf}(ctx_d^{bf}(s'))| \leq |\neg^{dnf}(ctx(s'))| + |\neg^{dnf}(ctx^{dt})| \leq {L_0}^{C_0} + (k-1)^{\lambda} \tag{7.9}$$

where $(k-1)$ represents the maximum number of literals per clause, and $|ctx^{dt}| = \lambda$ from Lemma 7.7. Note that, from the definition of operator $\neg^{dnf}$, $|\neg^{dnf}(c^{\vee})| \leq L^C$.

Now, Equations 7.8 and 7.9, and the definition of operator $\cap^{dnf}$, imply:

$$|ctx_d^{af}(s)| \leq |ctx_d^{bf}(s)| \times ({L_0}^{C_0} + (k-1)^{\lambda})$$

In the worst case scenario, service s could be dominated by all the remaining services, whose context conditions are eliminated similarly from $ctx_d^{bf}(s)$, resulting in:

$$|ctx_d^{af}(s)| \leq |ctx_d^{bf}(s)| \times ({L_0}^{C_0} + (k-1)^{\lambda})^{nc-1}$$

□

Lemma 7.9. *Given a service $s \in candidates^c(t)$, the inter-task pruning can only decreases the size of this service's post-domination context condition $ctx_d^{af}(s)$. That is:*

$$|ctx_d^{ai}(s)| \leq |ctx_d^{af}(s)|$$

Proof. The proof is straightforward from the definition of inter-task pruning. □

7.6.4 Discussion

In addition to the number of candidate services n, number of tasks k, and pruning rates, the analysis provided above implies that the other key factors affecting the complexity of our correlation-aware service selection algorithm are: the original context complexity of services (in terms of C_0 and L_0), the number of quality combinations per dependent service X, the correlation percentage per dependent task *crlpcg*, and the number of dependent tasks *crltsk* (see Section 7.7 for an empirical evaluation of the effect of these factors).

Regarding the original context complexity and number of quality combinations, the most common case is for a dependent service to provide two quality combinations: a special offer if used with another particular service, and a default offer otherwise. That is, $X = 2$ and $C_0 = L_0 = 1$. Such a case results in λ always being equal to 1 (i.e. there is no increase in *context size* by the pre-domination context restriction step), and ω

being bounded by $min(1+crltsk, k-1)^{nc-1}$. The latter bound is because, in this case, each $ctx_d^{bf}(s')$ is comprised of one clause with at most $min(1+crltsk, k-1)$ literals.

Since the majority of services are often quality independent, the correlation percentage per dependent task tends to be relatively low, and thus so does the number of dependee services (services participating in quality correlations). As a result, a large proportion of services per task (non-dependee services) are restricted by the same context, ctx^{dt}, during the pre-domination context restriction step. Given this, when a service is dominated by a considerable number of services, most of the dominating services are likely to share the same additional context ctx^{dt}, which eventually leads to only their original context conditions participating in the increase of the dominated service's context complexity. To illustrate, consider a service $s \in candidates^c(t)$ dominated by d other services, $s_1, ..., s_d \in candidates^c(t)$, with $ctx^{dt}(s_1) = ... = ctx^{dt}(s_d)$. Consequently, $ctx_d^{bf}(s)$ should be restricted by context ctx', i.e. $ctx_d^{af}(s) = ctx_d^{bf}(s) \cap^{dnf} ctx'$, such that:

$$
\begin{aligned}
ctx' &= [\neg^{dnf}(ctx_d^{bf}(s_1))] \cap^{dnf} [\neg^{dnf}(ctx_d^{bf}(s_2))] \cap^{dnf} ... \cap^{dnf} [\neg^{dnf}(ctx_d^{bf}(s_d))] \\
&= [\neg^{dnf}(ctx(s_1) \cap^{dnf} ctx^{dt}(s_1))] \cap^{dnf} [\neg^{dnf}(ctx(s_2) \cap^{dnf} ctx^{dt}(s_2))] \\
&\quad \cap^{dnf} ... \cap^{dnf} [\neg^{dnf}(ctx(s_d) \cap^{dnf} ctx^{dt}(s_d))] \\
&= [\neg^{dnf}(ctx(s_1)) \vee \neg^{dnf}(ctx^{dt}(s_1))] \cap^{dnf} [\neg^{dnf}(ctx(s_2)) \vee \neg^{dnf}(ctx^{dt}(s_2))] \\
&\quad \cap^{dnf} ... \cap^{dnf} [\neg^{dnf}(ctx(s_d)) \vee \neg^{dnf}(ctx^{dt}(s_d))] \\
&= [(\neg^{dnf}(ctx(s_1))) \cap^{dnf} (\neg^{dnf}(ctx(s_2))) \cap^{dnf} ... \cap^{dnf} (\neg^{dnf}(ctx(s_d)))] \quad \vee \\
&\quad [\neg^{dnf}(ctx^{dt}(s_1))]
\end{aligned}
$$

In other words, the domination-based context growth factor ω, resulting from such dominating services, is bounded by $\prod_{i=1}^{d}(|\neg^{dnf} ctx(s_i)|) + |\neg^{dnf} ctx^{dt}(s_1)|$ rather than by

$\prod_{i=1}^{d}(|\neg^{dnf} ctx(s_i)| + |\neg^{dnf} ctx^{dt}(s_i)|)$. That is, by ${L_0}^{C_0 \times d} + (k-1)^{\lambda}$ rather than by $({L_0}^{C_0} + (k-1)^{\lambda})^d$. Here, if $ctx^{dt}(s) = ctx^{dt}(s_1)$, factor ω is reduced to $\prod_{i=1}^{d} |\neg^{dnf} ctx(s_i)|$ (i.e. $\omega \leq {L_0}^{C_0 \times d}$). Furthermore, when in such a case, at least one dominating service, s_i, has an unrestricted original context condition $ctx(s_i) = \text{ALL}$ (a very likely scenario), or there exist two dominating services, s_i and s_j, with complementary original context conditions $ctx(s_i) = \neg^{dnf} ctx(s_j)$ (e.g. s_i and s_j are the quality combinations offered by a dependent service in S), factor ω becomes 0, i.e. the dominated service s is eliminated entirely from the search space. Note that when $X = 2$ and $C_0 = L_0 = 1$, the first two boundaries of ω correspond to $(1 + crltsk)$ and 1, respectively.

7.7 Empirical Study

This section presents an experimental evaluation of our correlation-aware service selection, focusing on its performance, in terms of execution time, and gain in utility achieved by considering correlations among services. The candidate services of each task are generated randomly (each service is assumed to have 5 additional quality attributes). Request quality constraints and quality weights are also set to random values. When a quality attribute a of service $s \in S$ is dependent on other services, it is assigned two possible randomly-generated values: a correlation value v_{crl}, whose context condition $cond(s, a, v_{crl})$ is generated randomly; and a default value v_{df}, whose context condition $cond(s, a, v_{df})$ is set to $\neg^{dnf}(cond(s, a, v_{crl}))$. All experiments were conducted on a PC with Intel Core 2 Quad 3GHz CPU and 3.23GB RAM.

To assess the effectiveness of the proposed prior-selection pruning, the running time of the selection algorithm is compared in three cases: no prior-selection pruning, only domination pruning, and both domination and inter-task pruning. Here, the number of

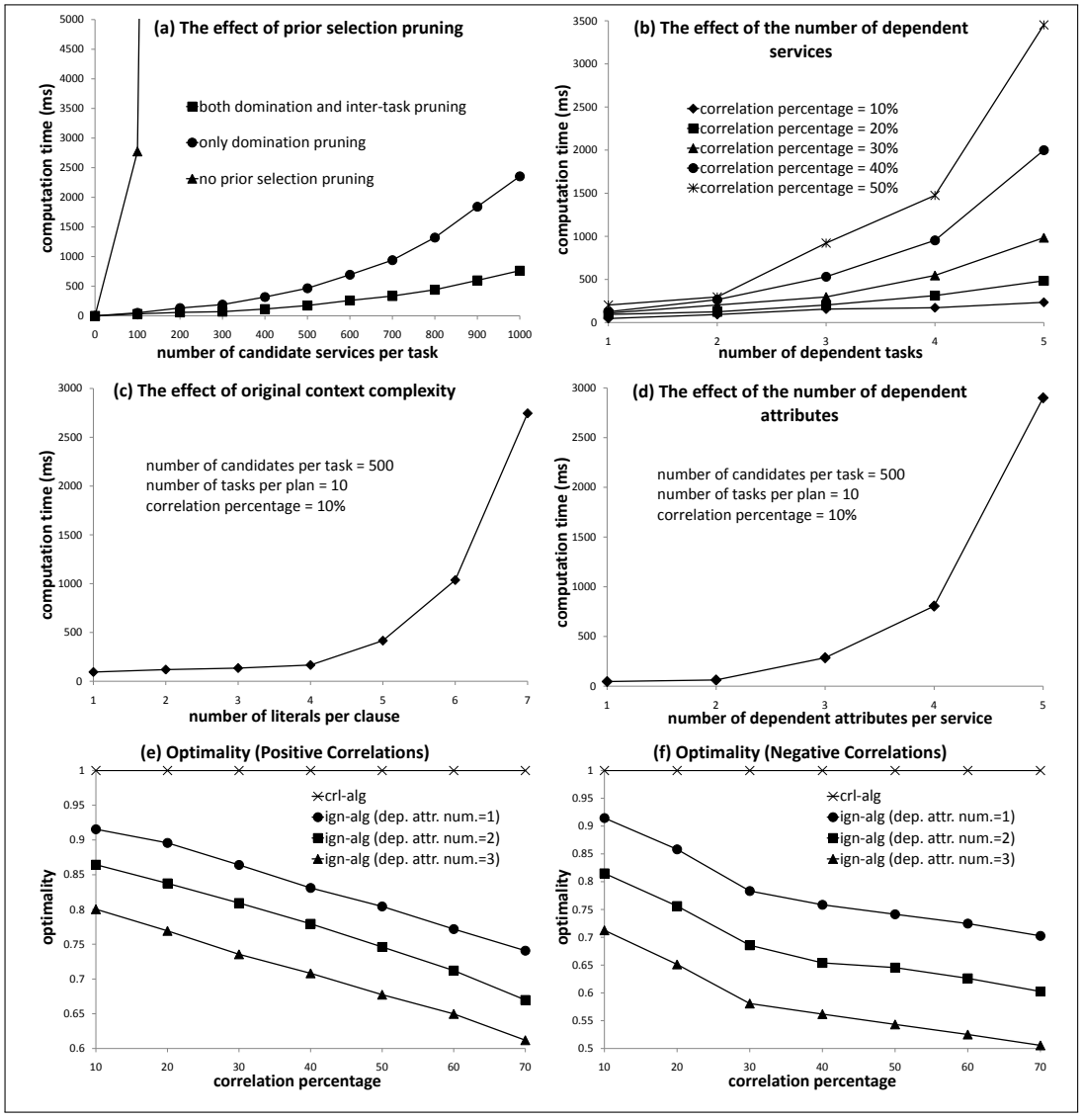

FIGURE 7.10: Evaluating the correlation-aware selection (all the results are averaged over multiple runs)

tasks per plan k, the correlation percentage per dependent task *crlpcg*, and the number of dependent tasks *crltsk*, are set to 10, 20%, and 4, respectively, while the number of candidate services per task n in the original candidate space is varied between 100 and 1000. The results (in Figure 7.10(a)) indicate that prior-selection pruning improves performance significantly. Moreover, applying inter-task pruning after domination pruning achieves further computation time reduction, which becomes more significant with the increasing number of dependent services (here, the number of dependent services per task for the minimum and maximum candidate number, is 20 and 200, respectively).

Figure 7.10(b) further shows the performance of our correlation-aware selection algorithm (when both domination and inter-task pruning are performed prior selection), with respect to the correlation percentage *crlpcg* and the number of dependent tasks *crltsk*. The number of tasks per plan, and the number of candidates per task, are fixed at 10 and 500, respectively. Clearly, as the number of dependent services grows (by increasing either the correlation percentage per task, or the number of dependent tasks), more services are replaced with multiple quality combinations, the context conditions are likely to increase in complexity, and fewer services will be pruned out *entirely* from the search space (for instance, due to being dependees for other services). These all lead to a corresponding increase in selection time. The negative effect on selection performance caused by original context complexity, and the number of quality combinations per service X, is highlighted in Figures 7.10(c) and 7.10(d), varying the number of literals per clause in the randomly generated context conditions, and the number of dependent attributes per dependent service, respectively.

To evaluate utility gain from correlation awareness, two algorithms were compared in terms of *solution optimality*: our correlation-aware selection *crl-alg*, and a correlation-ignorant selection *ign-alg* (i.e. the selection algorithm considering only the default values of the quality attributes, with their context conditions being equal to ALL). This

optimality is estimated as $\frac{utility_{c_{act}}}{utility_{c_{opt}}}$, where $utility_{c_{act}}$ and $utility_{c_{opt}}$ denote the actual utility achieved by the algorithm and the optimal utility, respectively. Since *ign-alg* ignores quality correlations, its estimated utility might not be accurate, and therefore the actual utility of its solution was re-calculated considering correlations. Figures 7.10(e) and 7.10(f) show the results, with the presence of positive (better than default) and negative (worse than default) quality correlations, respectively, varying correlation percentage between 10% and 70%, while fixing the candidate number per task at 100. As expected, *crl-alg* always produces the best possible solution, while *ign-alg*'s optimality decreases as more services become correlated because, the higher the correlation percentage, the more likely that *ign-alg* will neglect, in the case of Figure 7.10(e), much better solutions with positively correlated services, and select, in the case of Figure 7.10(f), non-optimal solutions containing negatively correlated services. Additionally, as can be observed, *ign-alg*'s optimality is negatively affected by the increasing number of dependent attributes per service (because more attributes have a higher impact on utility).

7.8 Conclusion

In this chapter, we have presented a correlation-aware service selection approach, capable of handling QoS dependencies among services, thus improving the quality of the composition produced by the selection process. For this purpose, the service model of Chapter 3 was modified, extending the attribute value function with multiple values per quality attribute; each with an associated applicability condition. To reduce selection cost, correlation-aware pruning techniques were introduced, which eliminate uninteresting compositions from the selection search space. Specifically, to enable dominance assessment among services, each service is first mapped to a number of representatives

corresponding to its possible quality value combinations. A request-based domination pruning of these representatives is then conducted at each task level, through appropriate restriction of their context conditions. This was followed by a further validation/restriction of the context conditions from a global perspective (i.e. taking all tasks into consideration), via inter-task pruning. An extension of our service selection algorithm, accommodating the correlation-driven modifications of the service space and the respective pruning, was also outlined.

To validate our approach, we have conducted a theoretical analysis of the complexity of the selection algorithm and pruning techniques (including their effect on context complexity), and further supported this with experimental results. Proofs of the optimality of our algorithm were also provided (see Appendix A for the theoretical proof).

Although the approach proposed addresses the correlation problem conceptually, it requires corresponding solutions from the practical implementation perspective. Particularly, suitable extensions to service description languages and invocation mechanisms are needed to express context-dependent quality attributes, and to make the current composition context explicit to a service on its invocation. Through a practical example in Chapter 8, we will suggest an appropriate extension to the description language of knowledge services, to accommodate their quality dependencies.

Chapter 8

Case Study: Learning Object Composition

8.1 Introduction

So far, all the experiments in this thesis were conducted on randomly-generated data. Although these experiments have demonstrated the effectiveness of the algorithms proposed, it is important to support our findings by showing the practical usefulness of the algorithms through a real-world example. For this purpose, the example adopted in this chapter is the dynamic generation of courses in the e-learning domain [109].

The rest of the chapter is organised as follows. The concept of e-learning is introduced in Section 8.2. Learning objects, key aspects of e-learning, are defined in Section 8.3, followed by the e-learning delivery platforms in Section 8.4. A mapping between our service selection model and the learning object composition problem is provided in Section 8.5, while a thorough evaluation of the proposed algorithms in such framework is carried out in Section 8.6. Finally, Section 8.7 concludes the chapter.

8.2 E-learning

Advances in information and communication technologies have enabled the emergence of electronic learning (*e-learning*): a new form of learning utilising these technologies to facilitate and enhance education and training. The educational content in such learning is usually delivered to learners through a wide range of electronic media, including the internet, intranet, CD ROM or DvD, etc [86]. E-learning can be useful in many domains and for different purposes, such as training the workers in an organisation, providing degree/certificate programs in an academic institution, or simply delivering any on-demand specific-purpose learning content (e.g. a course of study needed to perform a task or acquire a certain skill).

Unlike traditional classrooms, where teaching usually takes place at a specific time and specific place, e-learning eliminates the time and place constraints, and provides the learner with a more convenient learning experience, by offering a variety of possible learning models [39]: at any time, but from a specific place (e.g. accessing an online course from a specific learning centre); from anywhere, but at a specific time (e.g. synchronous virtual classrooms through video conferencing); or at any time, and from anywhere (e.g. asynchronous self-paced distance courses). Other advantages of e-learning include delivering personalised learning experiences that meet the needs and preferences of individual students, allowing students to learn at their own pace, providing interactive learning content enriched with graphics, simulations, animations, audio, video, etc., and reducing the cost for both the learner (e.g. due to decreased travel and accommodation expenses, and the ability to choose from a range of prices fitting different budgets) and the training organiser (e.g. due to cutting down the cost of classroom facilities, and the ability to reuse pre-existing quality learning material).

8.3 Learning Objects

Learning objects are one of the key aspects in e-learning, and are considered the building blocks for developing larger learning modules and courses. A learning object is a chunk of digital learning material usually designed to achieve a particular learning objective. Besides the actual multimedia content, a learning object might also contain practice and assessment items to support accomplishing its associated learning goal. Learning objects are usually self-contained and context-independent so that they can be reused in multiple instructional contexts. In fact, reusability is one of the main goals of learning objects (instead of continuous several-hour content chunks), since building quality learning material from scratch can be difficult, expensive and time consuming. Hence, by allowing the learning resources to be reused, whole courses can be assembled from pre-existing learning objects (possibly developed by others), resulting in more effective and economic learning.

8.3.1 Learning Object Metadata

To improve the reusability of learning objects, they need to be accessible. This is usually achieved by making the learning objects available through *learning object repositories*, and associating them with descriptive *metadata* (data about data) that facilitate their storage and retrieval. For this purpose, several metadata standards, such as Dublin Core [116] and IEEE Learning Object Metadata (LOM) [62], have emerged. For example, the IEEE LOM is an internationally recognised standard for describing learning objects. It offers a hierarchical data model (usually in XML format) that groups the characteristics of learning objects into the following nine categories.

- The *General* category provides general information about the learning object, such as title, language, description, keywords, aggregation level, etc.

- The *Lifecycle* category provides information about the current status and history of the learning object.
- The *Meta-Metadata* category provides information about the metadata instance itself, such as language, schema, creators, etc.
- The *Technical* category provides information about the technical properties of the learning object, such as format, size, duration, URL location, hardware/software requirements, etc.
- The *Educational* category provides information about the pedagogical properties of the learning object, such as interactivity type, interactivity level, difficulty, typical learning time, etc.
- The *Rights* category provides information about the conditions that constrain the use of the learning object, such as cost and copyright.
- The *Relation* category provides information about the relationship between this learning object and other learning objects.
- The *Annotation* category provides information about the educational comments created for the learning object, such as comment creator, date, and description.
- The *Classification* category describes the learning object according to a particular classification system.

8.3.2 Learning Object Content Models

Although a general agreement exists that a learning object should be reusable, there is a lot of debate regarding its granularity level, with many authors providing different suggestions for identifying the best size of the learning object, e.g. in terms of

the learning time, physical components (number of atomic assets), logical (pedagogical) structure, etc. As a result, learning objects can be available at a vast variety of granularities, i.e. they can range from single images or text fragments to complete modules or courses [48]. Elementary media items as well as large content units, however, may not be the best choices for learning object granularity. This is because, the former are highly decontextualised and thus, when reused, are very difficult to compose into instructionally meaningful units, while the latter's potential for reuse is low (learning object reusability usually decreases with its increasing size). Hence, the most appropriate granularity level for learning objects lies somewhere between these two extremes, but is difficult to specify. To address this difficulty, several content models have been proposed, aiming at providing an explicit description of the different levels of components comprising learning content, and their hierarchical structure within this content, thus enabling a multi-level disaggregation of learning resources into their components which, consequently, can be accessed and reused separately as required. The most widely adopted content model is the SCORM content aggregation model[1]. Other examples of content models include [20] the Learnativity content model, and Cisco RLO/RIO model.

SCORM Content Model. SCORM (sharable content object reference model) is a set of specifications and standards intended to produce accessible, reusable, and interoperable learning content (i.e. content that can be shared and exchanged across different delivery platforms). SCORM includes a content aggregation model that decomposes the learning material into assets, sharable content objects (SCOs), and content organisations. An asset is the simplest form of learning resource, such as text, image, audio, HTML fragment, etc. Assets can be combined together to form larger assets. A SCO is a collection of one or more assets, and represents the smallest learning unit that can be tracked by a learning management system (see Section 8.4) through the

[1] `http://www.adlnet.gov/scorm`

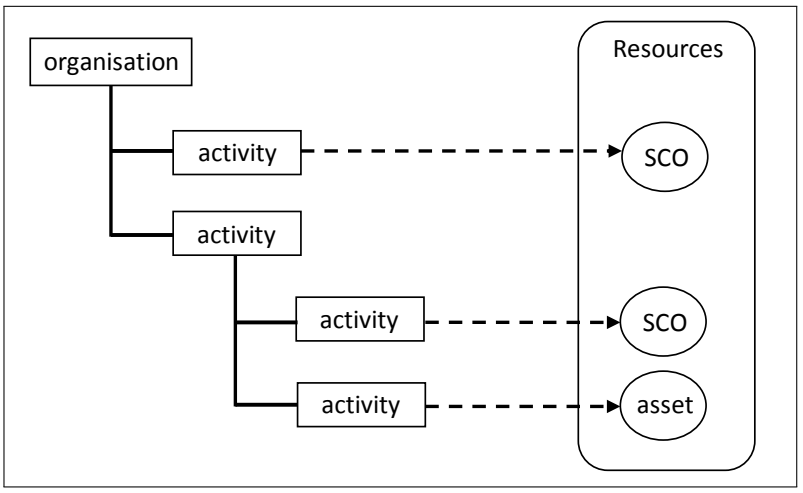

FIGURE 8.1: SCORM Content Organisation

use of SCORM runtime environment. SCOs should be context-independent so that they can be reused in various learning scenarios to achieve different goals. A content organisation is a map defining the logical structure of the learning content through a hierarchy of instructional units called activities (see Figure 8.1). Each activity can be further decomposed into sub-activities, with the leaf activities being associated with launchable learning resources (i.e. assets or SCOs). To control the order in which the learning content is viewed at run time, sequencing and navigation rules, which are interpreted by any SCORM-conformant learning management system, can be added to activities. All SCORM content model components (i.e. assets, SCOs, activities, and content organisations) can be described with metadata to facilitate their discovery and reuse.

8.4 Learning Delivery Platforms

In order to facilitate e-learning, a number of different educational computer applications, offering various functionalities and features, have emerged. Of the best known ones are learning management systems and learning content management systems.

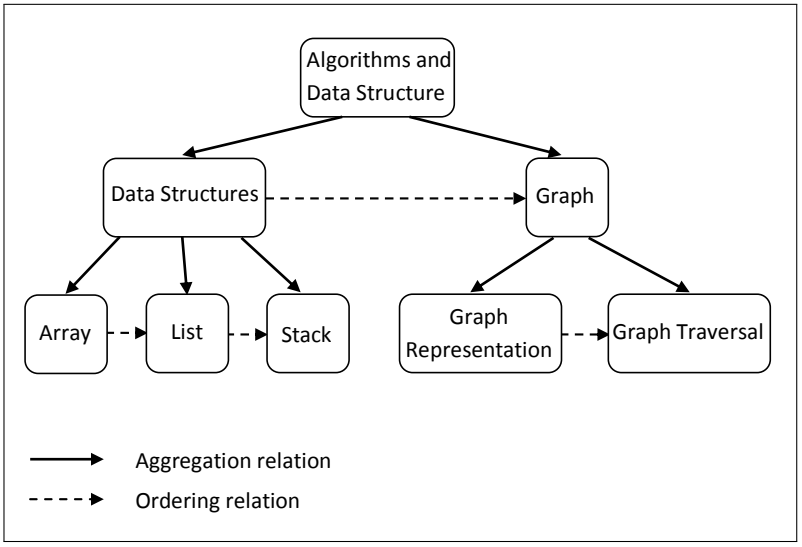

FIGURE 8.2: Part of the planning knowledge for the algorithms and data structure domain

A learning management system (LMS) is a software application that tackles all aspects of the *learning process*. Its main functionalities include delivering content to learners, handling registration and other administration tasks, assessing learner skills and tracking their progress, and providing reports.

A learning content management system (LCMS), on the other hand, is content-centric. That is, it is a system that handles the aspects of creating, storing, reusing, and delivering *learning content*, usually in the form of learning objects. Typically, it integrates a learning object repository (for storing and managing learning objects), content delivery tools, and automated authoring tools that allow content authors to create and upload new learning objects, or aggregate existing ones into larger units of instruction.

The functionalities of LMS and LCMS are somewhat complementary, and thus can be integrated to provide more comprehensive management of the learning process.

8.5 Learning Object Composition

Our evaluation is conducted on the learning object composition domain, where the goal is to fulfill a particular learning objective by automatically composing existing learning objects into a meaningful learning experience, taking learner (user) preferences and constraints into consideration [49]. A detailed mapping between our service selection model and the learning object composition problem is provided next.

8.5.1 Planning Knowledge Model

The planning knowledge for the learning object composition problem is a hierarchical representation of the domain knowledge, where the tasks are mapped to the concepts of the subject domain, while the aggregation and ordering relations can be interpreted as *has-part* and *has-prerequisite* relations, respectively, between these concepts. The relationship A *has-part* B indicates that concept B is a sub-concept of concept A, and thus learning concept A should include learning concept B. The relationship A *has-prerequisite* B means that, in order to understand concept A, it is necessary to have knowledge about concept B. Hence, concept B should be learned before concept A. Figure 8.2 shows part of such planning knowledge for the Algorithms and Data Structures domain. According to the semantics of the aggregation relation, acquiring concept *Basic Data Types*, for example, should incorporate acquiring the sub-concepts *Array*, *List* and *Stack*. Similarly, the ordering constraint between concepts *Graph Representation* and *Graph Traversal* implies that understanding the latter requires knowledge of the former, and thus *Graph Representation* should be presented to the learner first. The elements $(T, Taggr, root_t, graph_t)$ of this example planning knowledge are provided in Figure 8.3.

$$
\begin{aligned}
T = \{\ & \text{Algorithms and Data Structure}, \text{Basic Data Types},\\
& \text{Graph}, \text{Array}, \text{List}, \text{Stack}, \text{Graph Representation},\\
& \text{Graph Traversal}\ \}\\
Taggr = \{\ & \langle \text{Algorithms and Data Structure}, \text{Basic Data Types}\rangle,\\
& \langle \text{Algorithms and Data Structure}, \text{Graph}\rangle,\\
& \langle \text{Basic Data Types}, \text{Array}\rangle,\\
& \langle \text{Basic Data Types}, \text{List}\rangle,\\
& \langle \text{Basic Data Types}, \text{Stack}\rangle,\\
& \langle \text{Graph}, \text{Graph Traversal}\rangle,\\
& \langle \text{Graph}, \text{Graph Representation}\rangle\ \}\\
root_t = \ & \text{Algorithms and Data Structure}\\
graph_t(\text{Algorithms and Data Structure}) = \{\ & (\{\text{Basic Data Types}, \text{Graph}\},\\
& \{\langle \text{Basic Data Types}, \text{Graph}\rangle\})\}\\
graph_t(\text{Basic Data Types}) = \{\ & (\{\text{Array}, \text{List}, \text{Stack}\}, \{\langle \text{Array}, \text{List}\rangle, \langle \text{List}, \text{Stack}\rangle\})\ \}\\
graph_t(\text{Graph}) = \{\ & (\{\text{Graph Representation}, \text{Graph Traversal}\},\\
& \{\langle \text{Graph Representation}, \text{Graph Traversal}\rangle\})\ \}
\end{aligned}
$$

FIGURE 8.3: The elements of the planning knowledge shown in Figure 8.2

Note that the above projection of the planning knowledge on the problem of learning object composition abstracts out some of the problem's complexities unnecessary for the purpose of our evaluation. More specifically, the following assumptions are made.

Assumption 1. All the learning objects adopted for the purpose of assembling learning material are self-contained units of instruction achieving a particular learning objective (i.e. presenting a particular concept in a pedagogically meaningful way). Although this assumption neglects relevant learning objects with lower granularity level, such as individual definitions, examples, explanations, etc., it avoids going into the detail of instructional theories that are necessary to dynamically compose such finer-granularity objects into meaningful learning units, but are not the focus of this case study. This pedagogical dimension, however, can be easily incorporated into the planning knowledge, as depicted in Figure 8.4, where the instructional tasks corresponding to the hierarchical instructional method of teaching a concept are added as leaf nodes to the hierarchy, thus allowing the reuse of smaller sized learning objects (i.e. objects covering only a particular pedagogical aspect of a concept rather than its full pedagogical presentation) in the course generation process.

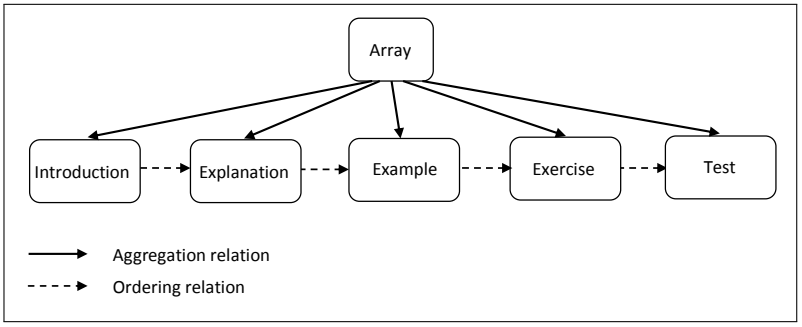

FIGURE 8.4: The addition of instructional tasks as sub-concepts to concept *Array*

Assumption 2. The sub-concepts of any non-leaf concept are totally ordered according to the *has-prerequisite* relation, i.e. for any two sub-concepts A and B of concept C, either A *has-prerequisite* B or B *has-prerequisite* A. The motivation behind this assumption is as follows. Although, in reality, the sub-concepts (of a particular parent concept) can be unrelated (i.e. *has-prerequisite* is only a *partial* order relation), a total order must be defined when presenting these sub-concepts to the learner, since the learner can only view one concept at a time (i.e. the learner cannot learn two different concepts simultaneously). This results in $n!$ possible decomposition graphs for each parent concept with n unrelated sub-concepts, each representing an alternative sequencing of these sub-concepts. Considering all such sequencing possibilities during the selection process will, however, only cause an additional overhead without any gain in information, since they all produce the same solution. Hence, to eliminate such plan redundancy, only one representative sequence is kept and considered a total order on the sub-concepts (this representative sequence can be chosen either randomly or according to the learner preferences). For example, in the planning knowledge of Figure 8.2, neither of the sub-concepts *Array* and *List* requires knowledge of the other (but both are prerequisites for *Stack*), leading to two alternative decompositions of concept *Basic Data Types*: *Array*, *List*, *Stack*; and *List*, *Array*, *Stack*. Yet, since changing the order of sub-concepts *Array* and *List* does not affect their optimal instantiation during selection, only one representative decomposition need be kept (as shown in Figure 8.2).

Assumption 3. The learner has the background necessary to understand the goal concept and all its sub-concepts, i.e. any prerequisite concept outside the hierarchy of the goal concept, is included in the learner's prior knowledge.

8.5.2 Quality Model

The quality of service attributes in the case of learning objects (the services in this case study), are the metadata elements usually associated with these objects to describe their properties, which we assume are compliant with IEEE LOM (the most widely used leaning object metadata standard). Due to the large number of elements provided by IEEE LOM, we only select a subset that we consider most relevant for the purpose of course generation, i.e., we choose the elements that allow learners to specify the properties of their preferred leaning experiences. Table 8.1 shows the selected metadata fields, along with their description, and value space. Notice that some metadata fields can be repeated more than once within a metadata instance (field R in Table 8.1), e.g. the language element can occur multiple times to indicate that the learning object is written in multiple languages.

The metadata elements of Table 8.1 (i.e. the quality attributes $NAME_a$), can be divided into three groups: *numerical*, *nominal categorical*, and *ordinal categorical*.

The *numerical* attribute group includes *Size*, *Duration*, *Typical Learning Time*, and *Cost*, all of which have a numerical domain, i.e.

$$dom_a(\text{Size}) \subset \mathbb{R}^+$$

$$dom_a(\text{Duration}) \subset \mathbb{R}^+$$

$$dom_a(\text{TypicalLearningTime}) \subset \mathbb{R}^+$$

$$dom_a(\text{Cost}) \subset \mathbb{R}^+$$

Field name	**Field representation in IEEE LOM**	**Field description**	**Field value**	**R**
Language	General.Language	The language of the learning object	An ISO language code (e.g. 'en-GB'), or value *none* for non-textual learning objects	yes
Size	Technical.Size	The physical size of the learning object (in bytes)	A string representing a decimal value	no
Duration	Technical.Duration.Datetime	The time required for running a continuous learning object (e.g. a video)	A string representing an ISO-compatible date	no
Interactivity Type	Educational.InteractivityType	The learning style that the learning object supports	One of the following: Active; Expositive; or Mixed	no
Semantic Density	Educational.SemanticDensity	How concise the learning object is	One of the following: very low; low; medium; high; or very high	no
Context	Educational.Context	The intended environment of the learning object	One of the following: Primary Education; Secondary Education; Higher Education; University First Cycle; University Second Cycle; University Postgrade; Technical School First Cycle; Technical School Second Cycle; Professional Formation; Continuous Formation; or Vocational Training	yes
Difficulty	Educational.Difficulty	How difficult the learning object is for the target audience	One of the following: very easy; easy; medium; difficult; or very difficult	no
Typical Learning Time	Educational.TypicalLearning Time. Datetime	The time required for understanding the content of the learning object	A string representing an ISO-compatible date	no
Cost	Rights.Description with Rights.Cost=yes; or Rights.Cost=no (in case of a free learning object)	The payment required for using the learning object	A string representing the cost	no

TABLE 8.1: The IEEE LOM metadata fields selected for the quality model

Each attribute in this group is associated with either an increasing or a decreasing direction, dir_a, depending on learner preferences (e.g. some learners prefer shorter learning experiences, while others prefer longer, more comprehensive ones).

The *nominal categorical* attribute group includes *Language* and *Context*, the domains of which are as follows.

$$dom_a(\text{Language}) = 2^L \quad \text{where } L = \{l \mid l \text{ is a language code}\}$$
$$dom_a(\text{Context}) = 2^C \quad \text{where } C = \{\text{Primary Education}, ..., \text{Vocational Training}\}$$

Since no ordering exists between the categories instantiating *Language* and *Context*, no direction is assigned to these attributes, i.e. $dir_a(\text{Language}) = dir_a(\text{Context}) = \text{none}$.

Finally, the *ordinal categorical* attribute group includes *Interactivity Type*, *Semantic Density*, and *Difficulty*. The possible values (categories) of these attributes are naturally ordered, and hence these attributes can be easily transformed to numerical by assigning representative scores to their values, as shown in Table 8.2. More specifically, the numerical domains of *Interactivity Type*, *Semantic Density*, and *Difficulty*, can be defined as follows.

$$dom_a(\text{InteractivityType}) = [1, 3]$$
$$dom_a(\text{SemanticDensity}) = [1, 5]$$
$$dom_a(\text{Difficulty}) = [1, 5]$$

Note that the domains of the above attributes are considered continuous (instead of discrete), to accommodate their possible values for learning object compositions (see Section 8.5.4). Similarly to the numerical group, each ordinal categorical attribute is associated with a *learner-defined* direction (either increasing or decreasing).

Interactivity Type	**Semantic Density**	**Difficulty**	**Numerical Values**
Expositive	very low	very easy	1
Mixed	low	easy	2
Active	medium	medium	3
	high	difficult	4
	very high	very difficult	5

TABLE 8.2: Numerical values for Interactivity Type, Semantic Density, and Difficulty attributes

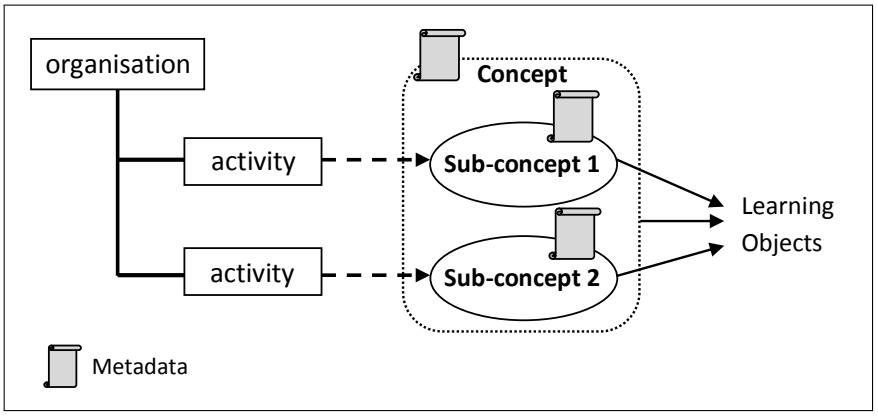

FIGURE 8.5: Nested Learning Objects in a SCORM Package

8.5.3 Service Model

The space of all learning objects (available through learning object repositories), along with their descriptive metadata, represent the service model $(S, name_s, value_s, func_s)$, where the semantics of the elements is as follows.

S is the set of all available learning objects, i.e. each learning object is considered a service offered by the learning object developer. Here, by learning object, we refer to any learning unit made available for reuse, and assume all such units are packaged according to the SCORM standard. Note that, if the logical components of a reusable learning unit are accessible (i.e. defined explicitly through SCORM activities), these components are also regarded as individual learning objects (see Figure 8.5), and thus can be reused in different learning contexts.

```
<general >
    <language>  en-GB  </language>
</general>
<technical>
    <size>  1000  </size>
    <duration>  <datetime> 00:00:00 </datetime>  </duration>
</technical>
<educational>
    <interactivitytype> <value> mixed </value> </interactivitytype>
    <semanticdensity> <value> high </value> </ semanticdensity >
    <context> <value> higher education </value> </ context >
    <difficulty> <value> easy </value> </ difficulty >
    <typicallearningtime>  <datetime> 00:30:00 </datetime>  </ typicallearningtime >
</educational>
<rights>
    <cost> <value> no  </value> </cost>
</rights>
```

FIGURE 8.6: An example of an IEEE LOM metadata instance

$name_s$ identifies the metadata fields available for each learning object. In this case study, we assume that, for each learning object, *all* the metadata fields defined by set $NAME_a$ (i.e. the metadata fields of Table 8.1) are instantiated (filled), i.e. $\forall s \in S, name_s(s) = NAME_a$.

$value_s$ specifies the characteristics offered by learning objects (i.e. their values for the metadata elements in $NAME_a$). These characteristics are extracted from the IEEE LOM metadata instance associated with each learning object, an example of which is showed in Figure 8.6.

$func_s$ defines the topic covered by each learning object (i.e. the concept(s) each learning object is instructing). Normally, such topic information can be derived from the *title*, *keyword*, and *description* fields, provided in the general category of IEEE LOM to describe the content of learning objects. However, to avoid complicated topic matching functions when identifying the relevant learning objects for tasks (not the focus of this

case study), we assume learning objects are described using the same domain concepts of the planning knowledge hierarchy. In other words, a learning object is considered candidate for a task if the concept describing the learning object in the *keyword* element matches the task's concept.

8.5.4 Composite Service Quality Model

The values of the quality attributes (the metadata elements of Table 8.1) for a composition of learning objects $lo_1 lo_2 ... lo_n$, are calculated by aggregating the respective values of the component learning objects, as follows.

$$value_c(lo_1 lo_2 ... lo_n, \text{Language}) = \bigcup_i value_s(lo_i, \text{Language})$$

$$value_c(lo_1 lo_2 ... lo_n, \text{Size}) = \sum_i value_s(lo_i, \text{Size})$$

$$value_c(lo_1 lo_2 ... lo_n, \text{Duration}) = \sum_i value_s(lo_i, \text{Duration})$$

$$value_c(lo_1 lo_2 ... lo_n, \text{InteractivityType}) = \frac{1}{n} \sum_i value_s(lo_i, \text{InteractivityType})$$

$$value_c(lo_1 lo_2 ... lo_n, \text{SemanticDensity}) = \frac{1}{n} \sum_i value_s(lo_i, \text{SemanticDensity})$$

$$value_c(lo_1 lo_2 ... lo_n, \text{Context}) = \bigcup_i value_s(lo_i, \text{Context})$$

$$value_c(lo_1 lo_2 ... lo_n, \text{Difficulty}) = \frac{1}{n} \sum_i value_s(lo_i, \text{Difficulty})$$

$$value_c(lo_1 lo_2 ... lo_n, \text{TypicalLearningTime}) = \sum_i value_s(lo_i, \text{TypicalLearningTime})$$

$$value_c(lo_1 lo_2 ... lo_n, \text{Cost}) = \sum_i value_s(lo_i, \text{Cost})$$

8.5.5 Request Model

A composition request, $(task_r, const_r, weight_r)$, describes the learning experience (the course) of interest to the learner, with the semantics of the components being as follows.

The requested task $task_r$, represents the learning goal, i.e. the concept the learner wants to acquire (learn about).

The quality constraints $const_r$, are the learner's constraints regarding course properties, i.e. constraints on the values of the metadata elements of the course. For nominal categorical attributes (language and context), these constraints specify the set of acceptable values, while in the case of numerical attributes (the remaining attributes), the constraints represent upper or lower bounds on the acceptable values, depending on the attribute direction (i.e. upper bounds for decreasing attributes, and lower bounds for increasing). For example, constraint '$const_r$(InteractivityType) = mixed' in the case where the direction is increasing, indicates that the learning mode supported by the course should be *at least* mixed (i.e. values *mixed* and *active* are both acceptable). However, if attribute *interactivity type* is associated with a decreasing direction, the same constraint indicates that the learning style should be *at most* mixed (i.e. the allowed values are *mixed* and *expositive*). Furthermore, the constraints on the selected metadata elements can be classified into local constraints and global constraints. The former, which include language and context constraints, can be met by filtering out, from each task individually, all the unsatisfactory learning objects. The latter, on the other hand, comprise the constraints on the remaining metadata elements, and should be checked against the *aggregated* values of the learning objects in a composition. Notice that constraints on ordinal categorical attributes are translated to numerical ranges, as shown in Figure 8.7. For example, constraint '$const_r$(InteractivityType) = mixed' is interpreted as '*interactivity type* should be more than 1.5' (if *interactivity type* is

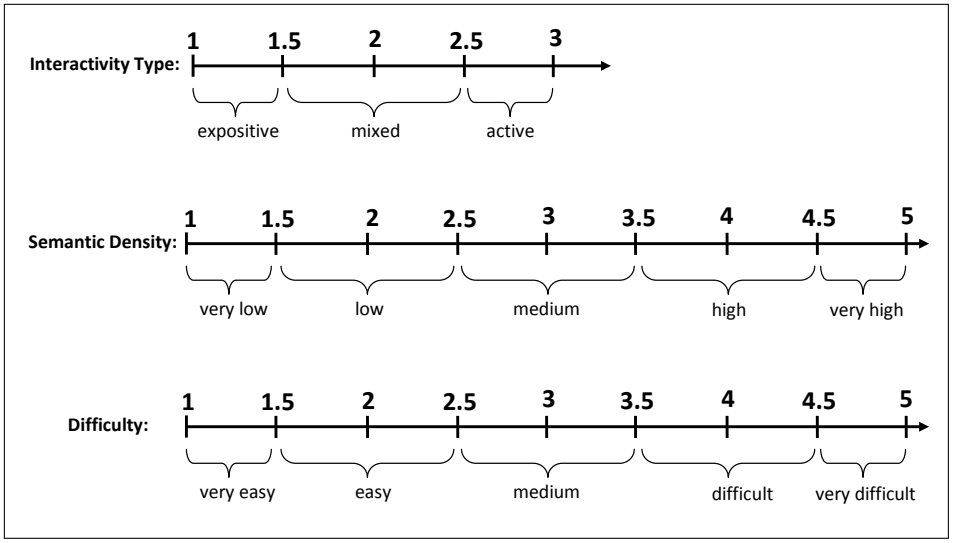

FIGURE 8.7: Numerical ranges for ordinal categorical constraints

increasing), and as '*interactivity type* should be less than 2.5' (if *interactivity type* is decreasing).

The quality weights $weight_r$, represent the relative importance of each *numeric* metadata element (i.e. metadata element with an associated *direction*) for the learner. For instance, if the learner is equally interested in maximising interactivity, while minimising difficulty, learning time, and cost, regardless of the other characteristics, the weights should be assigned as follows.

$$
\begin{aligned}
weight_r(\text{Size}) &= 0 \\
weight_r(\text{Duration}) &= 0 \\
weight_r(\text{InteractivityType}) &= 0.25 \quad \text{with } dir_a(\text{InteractivityType}) = + \\
weight_r(\text{SemanticDensity}) &= 0 \\
weight_r(\text{Difficulty}) &= 0.25 \quad \text{with } dir_a(\text{Difficulty}) = - \\
weight_r(\text{TypicalLearningTime}) &= 0.25 \quad \text{with } dir_a(\text{TypicalLearningTime}) = - \\
weight_r(\text{Cost}) &= 0.25 \quad \text{with } dir_a(\text{Cost}) = -
\end{aligned}
$$

8.5.6 Correlations among learning objects

The values of some learning object characteristics (metadata elements) may not be fixed, but vary depending on the *learning context* in which the learning object is used. We can identify three such characteristics in the selected metadata elements: *typical learning time*, *difficulty*, and *cost*, with examples of cases where their values are context-dependent being provided below.

The *typical learning time* of learning object lo_1, presenting the concept of backtracking algorithms using the eight queens example, is 45 minutes. Part of this time, 15 minutes, is dedicated to understanding the example details (i.e. the data structure, solution representation, etc). Hence, if another learning object that precedes lo_1 in the learning context adopts the same explanatory example (e.g. learning object lo_2, illustrating the recursion concept through the eight queens problem), lo_1's learning time will be reduced to 30 minutes, since the learner will be already familiar with the example, and therefore can acquire the content of lo_1 much faster. The developer (or annotator) of lo_1 may be aware of lo_2, and thus can highlight the effect of their combination in lo_1's metadata, especially that recursion and backtracking concepts are often learned together.

Learning object lo_3, which presents the depth first graph traversal algorithm, is normally considered *difficult* to learn. This is due to the heavy use of mathematical notation in this object, which is defined on the graph representation structure, the adjacency list structure, for the purpose of explaining the algorithm. However, in the special case where graph representation methods, which usually precede graph traversal in the learning context, are presented by lo_4 (a learning object developed by the same author of lo_3), learning object lo_3 becomes of *medium difficulty*, since all its mathematical notation is properly explained through examples in lo_4, when presenting the adjacency list representation for graphs.

The *cost* charged for using learning object lo_5 (about the Queue Data Type), in an educational course (for instance, about Data Structures), is normally £20. However, this object's developer offers a £10 discount on its price (i.e. charges £10 instead of £20), if learning object lo_6 (also owned by the same developer) is the one selected for presenting the Stack Data Type in the same course.

In the IEEE LOM standard, relationships between the learning object under description and other learning objects can be described using the *Relation* element, which allows identification of the relationship kind (<relation>.<kind>) from a list of predefined vocabulary, and the target learning object (<relation>.<resource>). Despite being insufficient for expressing the quality correlations of interest among learning objects, it can easily be adapted to achieve this (i.e. to acquire the required expressive power), as follows: instead of referencing the target learning object by <relation>.<resource>, this element can be used to refer to a *correlation descriptor* resource (an XML document describing the quality correlations of the current learning object). The suggested structure (comprising elements) for such a correlation descriptor is detailed next.

<allcorrelations> Element

Description: This element is the root element of a correlation descriptor document.

Multiplicity: This element occurs once and only once within a correlation descriptor document.

Attributes: None.

Sub-elements:

- <correlation>

<correlation> Element

```
<correlation >
    <propertyname>  typical learning time  </ propertyname >
    <propertyvalue>  00:30:00  </ propertyvalue >
    <context>
        <clause>
            <literal  positive=true>
                <concept>  recursion  </concept>
                <range>
                    <resource>
                        <identifier> lo2 </ identifier >
                    </resource>
                </range>
            </literal>
        </clause>
    </ context >
</correlation>
```

FIGURE 8.8: <correlation> element example

Description: This element describes a quality correlation between the current learning object and (a) set(s) of other learning objects (the dependee objects), with respect to which the current object provides a specific special value for a particular property (see Figure 8.8 for an example).

Multiplicity: This element occurs 0 or more times within the <allcorrelations> element, depending on the number of quality correlations of the learning object.

Attributes: None.

Sub-elements:

- <propertyname>
- <propertyvalue>
- <context>

<propertyname> Element

Description: This element specifies the property (metadata element) concerned with the current correlation. It takes one of the following values: typical learning time, difficulty, or cost.

Multiplicity: This element occurs once and only once within the <correlation> element.

Attributes: None.

Sub-elements: None.

<propertyvalue> Element

Description: This element specifies the value that the current learning object delivers for property <correlation>.<propertyname> under the current correlation (i.e. when the correlation context <correlation>.<context> is satisfied).

Multiplicity: This element occurs once and only once within the <correlation> element.

Attributes: None.

Sub-elements: None.

<context> Element

Description: This element identifies the learning objects that need to be selected with the current learning object in order for the latter to deliver value <correlation>.<propertyvalue> for property <correlation>.<propertyname>.

Multiplicity: This element occurs once and only once within the <correlation> element.

Attributes: None.

Sub-elements:

- <clause>

<clause> Element

Description: This element describes a possible combination of the *dependee* learning objects that makes value <correlation>.<propertyvalue> of property <correlation>.<propertyname> applicable.

Multiplicity: This element occurs one or more times within the <context> element, with each occurrence representing an alternative combination of the *dependee* learning objects.

Attributes: None.

Sub-elements:

- <literal>

<literal> Element

Description: This element constrains the learning objects to be selected for a particular concept.

Multiplicity: This element occurs one or more times within the <clause> element, depending on the number of constrained concepts.

Attributes:

- positive. This attribute specifies whether the learning objects identified by <literal>.<range> should be selected (positive=true), or avoided (positive = false), for concept <literal>.<concept>.

Sub-elements:

- <concept>
- <range>

<concept> Element

Description: This element identifies the concept constrained by the current literal.

Multiplicity: This element occurs once and only once within the <literal> element.

Attributes: None.

Sub-elements: None.

<range> Element

Description: This element specifies the list of learning objects that should be selected (or avoided) for concept <literal>.<concept>.

Multiplicity: This element occurs once and only once within the <literal> element.

Attributes: None.

Sub-elements:

- <resource>

<resource> Element

This element identifies a particular learning object, and is similar to the <resource> sub-element of the original <relation> element.

8.6 Evaluation

Given the mapping presented in the previous section, we now show the experimental evaluation of the algorithms proposed throughout the thesis, applied in the domain of learning object composition.

8.6.1 Evaluation Setup

The planning knowledge hierarchy adopted in this case study concerns the *Algorithms and Data Structures* domain, and is depicted in Figure 8.9. This hierarchy is comprised of 49 concepts (tasks) distributed over 5 levels, with a maximum of 6 sub-concepts per each parent concept.

The LOM metadata instances of the candidate learning objects are collected using the ARIADNE harvesting tool[2], which allows the harvesting of metadata records from learning object repositories using the OAI-PMH protocol[3] (the Open Archives Initiative Protocol for Metadata Harvesting). In total, 36,956 metadata instances were obtained from 10 different repositories, as detailed in Table 8.3. To avoid going into details of semantic content similarity (not the focus of this case study), we assume all the learning objects collected are relevant to the domain of interest (Algorithms and Data Structures), and assign them to the hierarchy's concepts based on size and learning time (or randomly when size and learning time are not provided).

Despite their importance for the reuse of learning objects, LOM data elements, especially the educational elements, are often misused or not instantiated. This is confirmed by a recent study on the actual use of metadata [95], and our analysis of the

[2] `http://ariadne.cs.kuleuven.be/lomi/index.php/Harvesting_Metadata`
[3] `http://www.openarchives.org/OAI/openarchivesprotocol.html`

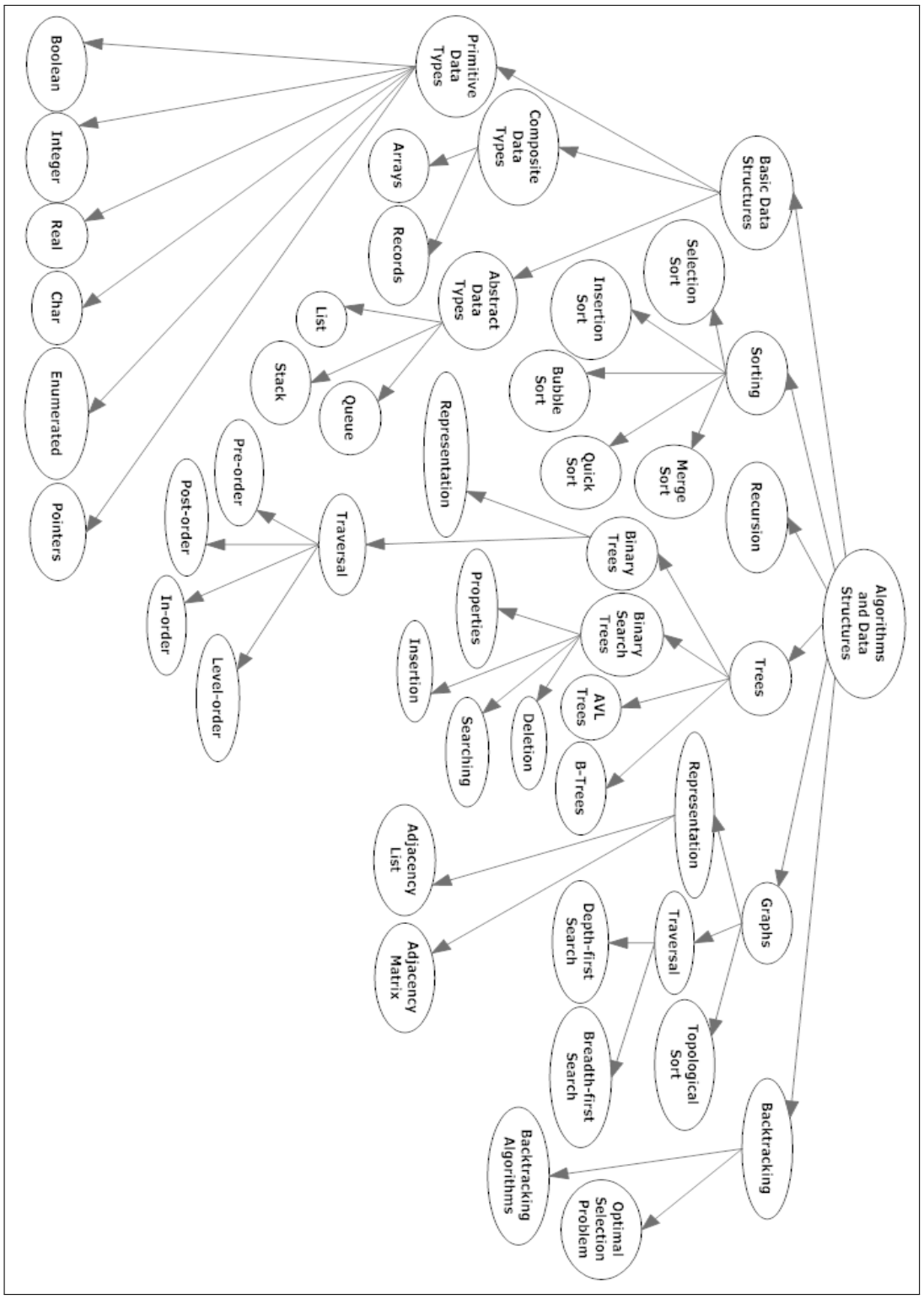

FIGURE 8.9: The Planning Knowledge Hierarchy used for Evaluation

LO Repository	Number of Metadata Instances Harvested
ait.opetaja.ee	4081
dum.rvp.cz	2170
oai.cndp.fr	1200
oai_lornet.org	24
oaicat.indire.it	16383
openlearn.open.ac.uk	634
repository-intralibrary.leedsmet.ac.uk	2328
rural.inclusion.eu	407
scam.kmr.se	7292
sodis.de.s.cp	2437

TABLE 8.3: Harvested Repositories

harvested metadata instances. Consequently, for our evaluation, we developed the following heuristics for filling in the missing values of the metadata elements of interest.

8.6.1.1 Language and Context

The language and context elements are very often filled by learning object annotators, and therefore no heuristic is required.

8.6.1.2 Interactivity Type

When the interactivity type of a learning object is not specified, it can be derived from the more usually provided learning resource type element. Table 8.4 shows the different values used for learning resource type in the harvested metadata instances, and identifies, for each of these values, the respective interactivity type according to IEEE LOM guidelines.

8.6.1.3 Semantic Density and Difficulty

If not provided, the semantic density and difficulty of a learning object are generated randomly from the sets of values $\{\text{very low}, \text{low}, \text{medium}, \text{high}, \text{very high}\}$ and

{very easy, easy, medium, difficult, very difficult}, respectively, based on a uniform probability distribution.

8.6.1.4 Size

In order to assign realistic size values to learning objects (when these values are absent in the metadata), we estimate for each domain concept, the minimum and maximum number of pages required to present this concept. These estimates are shown in Table B.1 of Appendix B. The resulting size ranges (given one byte per character and an average of 3000 characters per page), are then used to restrict the random generation of the missing size values for the candidate learning objects of each concept, assuming uniform probability distributions over these ranges. Note that, since the technical format of a learning object (i.e., text, image, audio, etc) has no effect on the values of the metadata elements used in this evaluation, only the textual format is considered here for simplicity.

8.6.1.5 Typical Learning Time

The time required to acquire a learning object usually depends on the amount of information presented by this object. Thus, when not specified, this time can be considered proportional to the learning object's size and semantic density, and calculated as:

$$\begin{aligned}\text{TypicalLearningTime} =&(\frac{\text{SemanticDensity} - \text{minSemanticDensity}}{\text{maxSemanticDensity} - \text{minSemanticDensity}} \times \frac{\text{Size} - \text{minSize}}{\text{maxSize} - \text{minSize}})\times \\ &(\text{maxLearningTime} - \text{minLearningTime}) + \text{minLearningTime}\end{aligned}$$

TABLE 8.4: Interactivity Type according to Learning Resource Type

Learning Resource Type	**Interactivity Type**	**Learning Resource Type**	**Interactivity Type**	**Learning Resource Type**	**Interactivity Type**
assessment	**Active**	presentation	**Expositive**	workbook	**Mixed**
drill and practice	**Active**	guide	**Expositive**	book	**Mixed**
educational game	**Active**	exploration	**Expositive**	book chapter	**Mixed**
simulation	**Active**	image	**Expositive**	edited book	**Mixed**
demonstration	**Active**	video	**Expositive**	course	**Mixed**
matchinteraction	**Active**	audio	**Expositive**	lecture	**Mixed**
choiceinteraction	**Active**	animation	**Expositive**	web page	**Mixed**
inlinechoiceinteraction	**Active**	narrative text	**Expositive**	case study	**Mixed**
extendedtextinteraction	**Active**	journal article	**Expositive**		
textentryinteraction	**Active**	scholarly text	**Expositive**		
associateinteraction	**Active**	conference contribution	**Expositive**		
hottextinteraction	**Active**	conference proceedings	**Expositive**		
orderinteraction	**Active**	electronic journal	**Expositive**		
gapmatchinteraction	**Active**	book review	**Expositive**		
hotspotinteraction	**Active**	report	**Expositive**		
sliderinteraction	**Active**	diagram	**Expositive**		
open activity	**Active**	figure	**Expositive**		
experiment	**Active**	slide	**Expositive**		
exercise	**Active**	booklet	**Expositive**		
quiz	**Active**	photograph	**Expositive**		
exam	**Active**	working or discussion paper	**Expositive**		
self assessment	**Active**	web publication	**Expositive**		
workshop	**Active**	thesis or dissertation	**Expositive**		
tutorial	**Active**	published abstract	**Expositive**		
problem statement	**Active**	audio recording	**Expositive**		
questionnaire	**Active**	graph	**Expositive**		
game	**Active**	text	**Expositive**		
x-stream assessment	**Active**	guide (advice sheets)	**Expositive**		
		lecture transcript	**Expositive**		

Here, *minSemanticDensity* and *maxSemanticDensity* correspond to *very low* and *very high* semantic densities, respectively. That is, $minSemanticDensity = 1$, and $maxSemanticDensity = 5$. $minLearningTime$ and $maxLearningTime$ are the minimum and maximum learning time estimates of the learning object's respective concept, defined based on domain knowledge (see Table B.1 of Appendex B). Similarly, *minSize* and *maxSize* represent the minimum and maximum concept size estimates. In the case where the typical learning time of the learning object is known, the above equation can be used to derive its size and semantic density (if not provided).

Since the typical learning time of a learning object is not necessarily deterministic in reality (in terms of its size and semantic density), this time is assigned randomly to a percentage α of learning objects with unknown learning time (e.g. $\alpha = 10\%$), from the uniform distribution on the interval $[\text{minLearningTime}, \text{maxLearningTime}]$.

8.6.1.6 Cost

When the price of a learning object is not provided, it is generated randomly from the interval $[0, \text{maxCost}]$, where *maxCost* is the maximum cost estimate for the learning object's respective concept (see Table B.1 of Appendix B). Since active learning objects are likely to be more expensive than passive ones, the probability distribution of the random variable representing cost should satisfy the following.

- If the learning object's interactivity type is *active*, higher values in interval $[0, \text{maxCost}]$ should receive higher probability, when randomly generating cost for this object.
- If the learning object's interactivity type is *passive*, lower values in interval $[0, \text{maxCost}]$ should receive higher probability, when randomly generating cost for this object.

- If the learning object's interactivity type is *mixed*, medium values in interval $[0, \text{maxCost}]$ should receive the highest probability, when randomly generating cost for this object.

The above could be achieved though appropriate parametrisation of the Beta Probability Distribution, according to interactivity type.

8.6.2 Static Selection of Learning Objects

As seen earlier, learning objects are published through learning object repositories, where hundreds of learning objects with different properties can be available for each concept in the planning knowledge hierarchy. This makes learning object composition a suitable domain for evaluating the usefulness of the pruning techniques presented in Chapter 4. This evaluation is detailed next.

8.6.2.1 Domination-based Pruning

This section evaluates the gain in performance achieved as a result of applying domination based pruning on learning objects prior to performing learning object selection. For this purpose, learning object selection time is compared in three cases: no domination pruning (the basic algorithm), request-independent domination pruning alone, and both request-independent and request-dependent domination pruning. Figure 8.10 shows the results, varying the percentage of the candidate leaning objects considered per concept between 20% and 100%, and averaging the execution time of each algorithm over 30 requests concerning the *Tree* goal concept. Each request has two global constraints (generated randomly), and a utility optimisation requirment with equal weights being assigned to all the global attributes (i.e. $\forall a, \quad weight_r(a) = 0.166$). As can be observed, both domination types significantly outperform the basic algorithm,

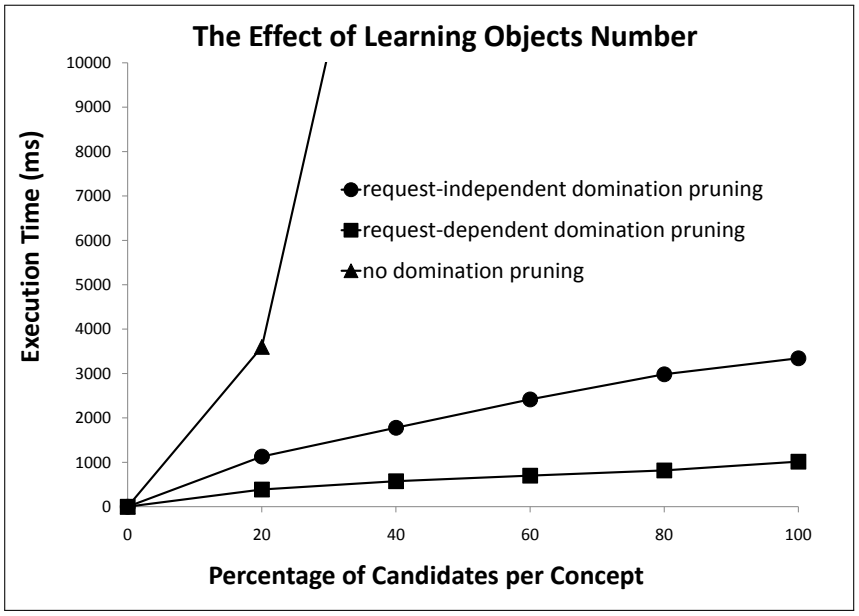

FIGURE 8.10: The effect of domination-based pruning

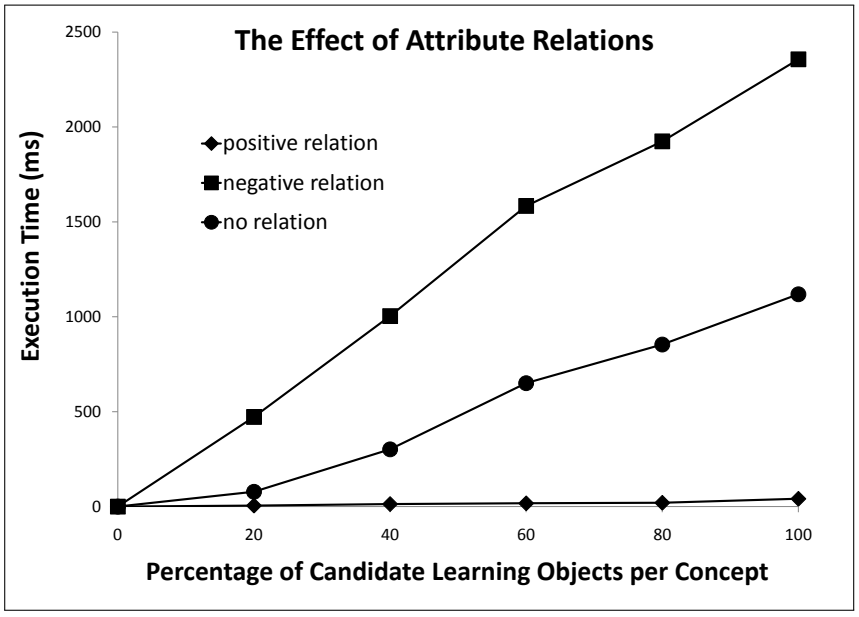

FIGURE 8.11: The effect of attribute relations on domination-based pruning

	Positive Relation	**Negative Relation**	**No Relation**
Goal Concept	Tree		
Constrained Elements	Typical Learning Time Size	Typical Learning Time Semantic Density	Semantic Density Size
Requested Directions	dir_a(TypicalLearningTime) = - dir_a(Size) = - dir_a(SemanticDensity) = +		
Optimisation Requirement	Minimise Cost		

TABLE 8.5: Requests Evaluated in Figure 8.11 (30 requests are generated per each relation type)

whose execution time increases dramatically as the number of learning objects per concept grows. Moreover, applying request-dependent pruning after request-independent pruning improves performance considerably, especially with the increasing number of learning objects.

Figure 8.11 further studies the performance of our request-dependent domination pruning, with respect to the relationships between constrained quality elements. Specifically, in the presence of positive relationships between the constrained elements (improving one results in improving others), more learning objects are eliminated due to being dominated by others, resulting in a corresponding positive effect on selection time. In contrast, when the constrained elements are negatively related (improving one leads to degrading others), more learning objects become incomparable, thus decreasing the pruning rate and consequently increasing selection complexity. Note that, since the learning time of a learning object in our evaluation is proportional to its size and semantic density, we simulate the positive and negative relations among attributes through appropriate settings of the directions requested for semantic density, size, and learning time (see Table 8.5). The other experimental settings are the same as previously.

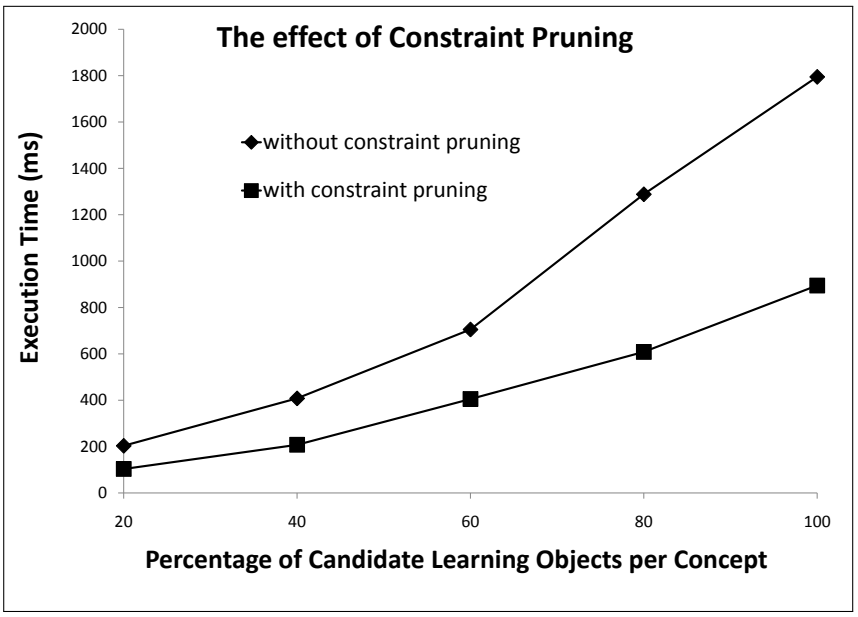

FIGURE 8.12: The effect of constraint-based pruning

8.6.2.2 Constraint-based Pruning

This section studies the improvement in execution time as a result of applying constraint-based pruning prior and during learning object selection. For this purpose, the running time of the selection algorithm is compared in two cases: without constraint-based pruning, and with constraint-based pruning. Again, the effect of the number of candidate learning objects is considered, with the experimental settings being the same as previously. The results in Figure 8.12 demonstrate the effectiveness of constraint-based pruning in selection time reduction. Intuitively, the gain in performance obtained is dependent on the strictness level of the imposed constraints, as depicted in Figure 8.13. This is because, in problems with overly strict constraints, more learning objects are pruned due to being unsatisfactory, as opposed to problems of lower strictness, where fewer learning objects are likely to violate the constraints.

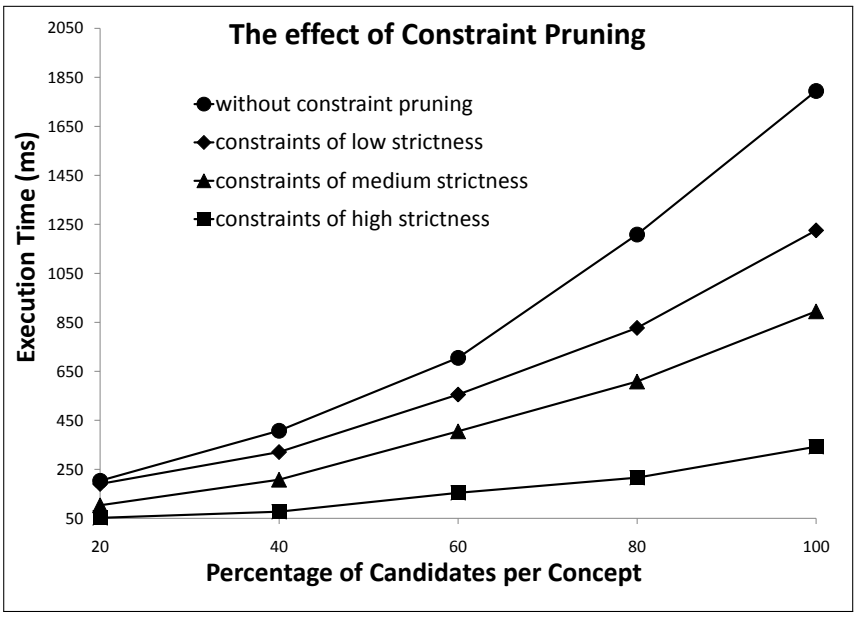

FIGURE 8.13: The effect of constraint strictness on constraint-based pruning

8.6.2.3 Plan-based Pruning

This section evaluates the gain in performance achieved as a result of applying plan-based pruning on the candidate composition plans (abstract plans), prior to performing learning object selection. For this purpose, the performance of the selection algorithm with no plan-based pruning is compared with that including plan-based pruning. Figure 8.14 shows the running times, varying the number of alternative abstract plans (note that the higher the level of the goal concept, the larger the number of alternative abstract plans). Like before, all the times reported are averaged over 30 randomly-generated requests, each comprised of two global constraints, and a utility optimisation requirement (where all the global attributes receive equal weights). The results demonstrate the effectiveness of plan-based pruning in computation time reduction, which increases with the increasing number of alternative composition plans.

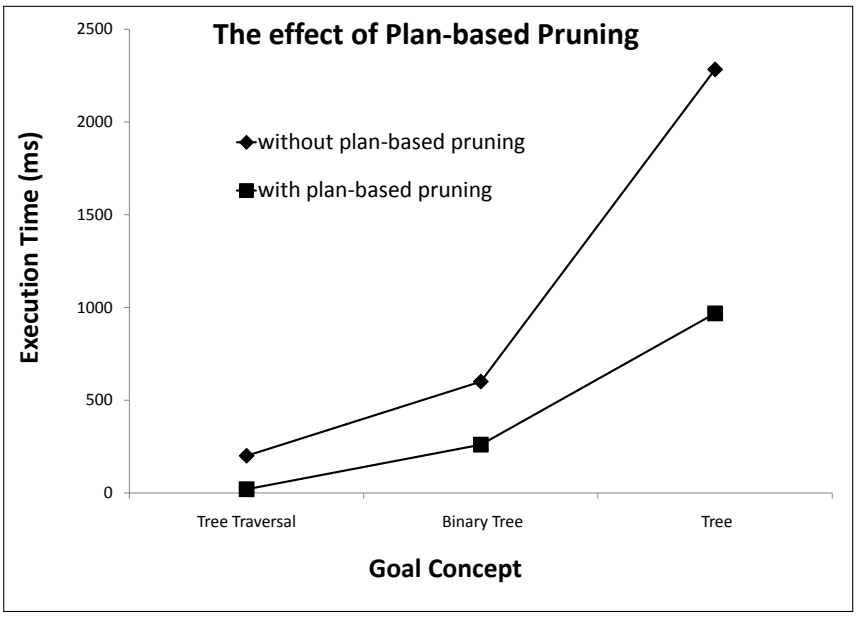

FIGURE 8.14: The effect of plan-based pruning

8.6.3 Reactive Selection of Learning Objects

Learning Object Repositories are not static, but dynamic in nature, where thousands of new learning objects may be made available every day, while existing learning objects may be removed or updated, at any time. In the latter case, the modification could occur either to the learning object itself (e.g. additional content is added to the learning object, correspondingly affecting its learning time, size, etc.), or only to its metadata instance (e.g. more accurate metadata becomes available for the learning object after an expert review or user feedback). Such changes to the available learning objects could have an impact on the optimality, satisfaction, and availability of the solution learning experience selected for the user, and hence should be accounted for during the selection process.

To test the effectiveness of our reactive selection in the learning object framework, we reproduce the same experiments performed in Section 5.5. The respective experimental results are shown in Figures 8.15-8.20, where similar observations regarding

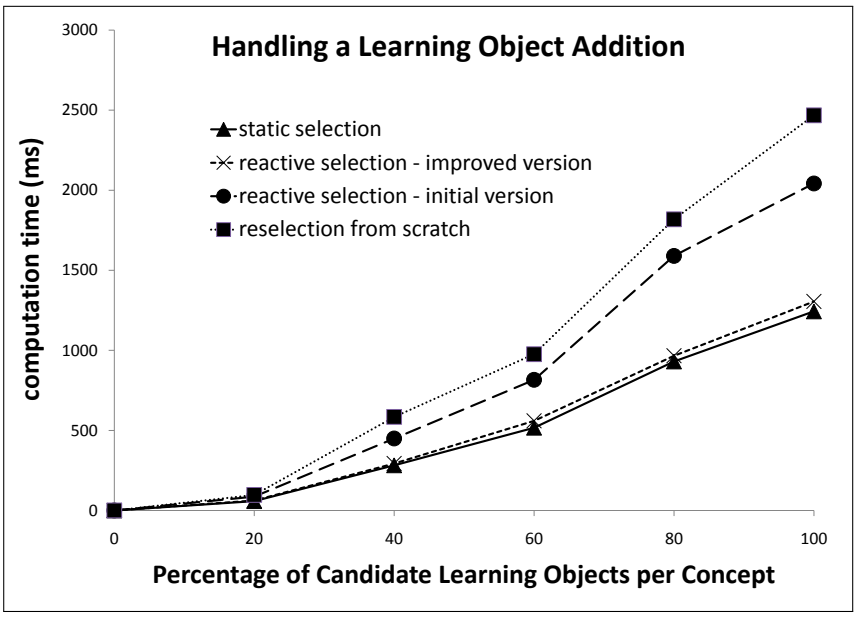

FIGURE 8.15: Addition of a learning object during selection

the efficiency and optimality of our reactive selection algorithm, can be made for the case of learning objects. Note that the results reported here are averaged over multiple requests (30 in Figures 8.15-8.19, and 300 in Figure 8.20). Each request consists of two randomly-generated global constraints and a randomly-generated optimisation requirement, with the goal concept being set to *Tree* (i.e. the search graph contains 7 abstract plans, with up to 11 tasks per plan). The percentage of candidate learning objects considered per task is varied between 20% and 100% in Figures 8.15-8.18, while fixed at 100% in Figures 8.19 and 8.20. For all changes, the task being processed when the change occurs is assumed to be task *B-Trees*, the last task in the topological order of the search graph (i.e. the worst case scenario), while change locations (the task and learning object affected by the change) are selected randomly. Changes concerning the addition/modification of a learning object are simulated by randomly generating a new metadata instance for this object, in accordance with the heuristics provided in Section 8.6.1.

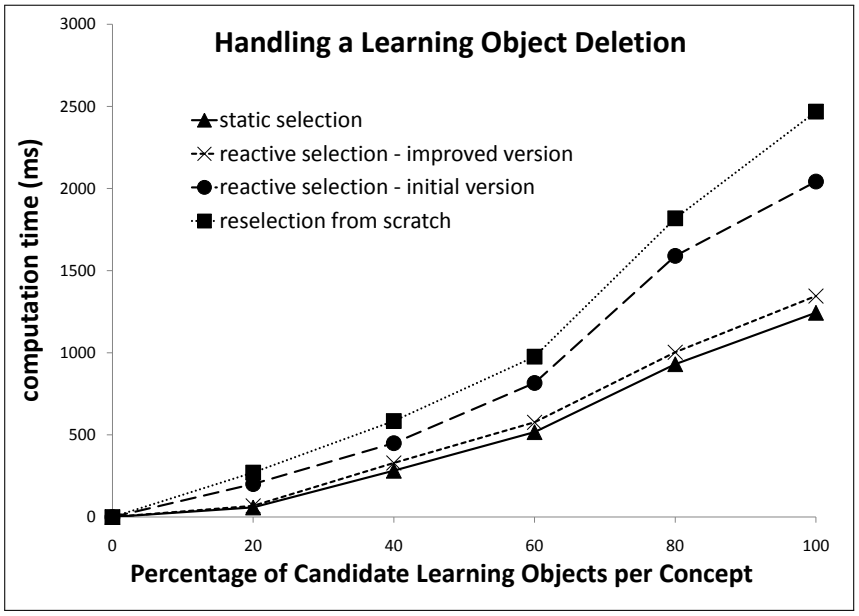

FIGURE 8.16: Deletion of a learning object during selection

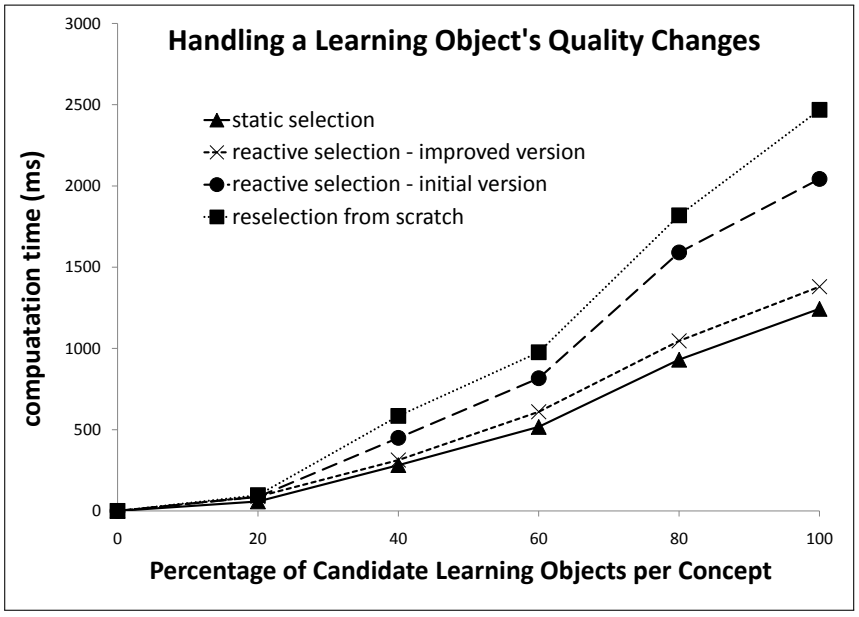

FIGURE 8.17: Changes in a learning object's qualities during selection

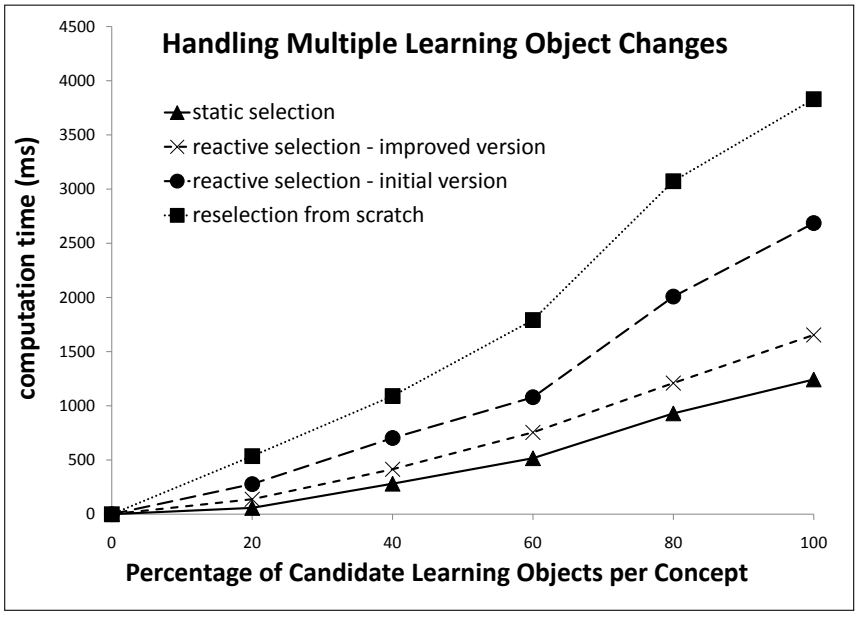

FIGURE 8.18: Multiple learning object changes during selection

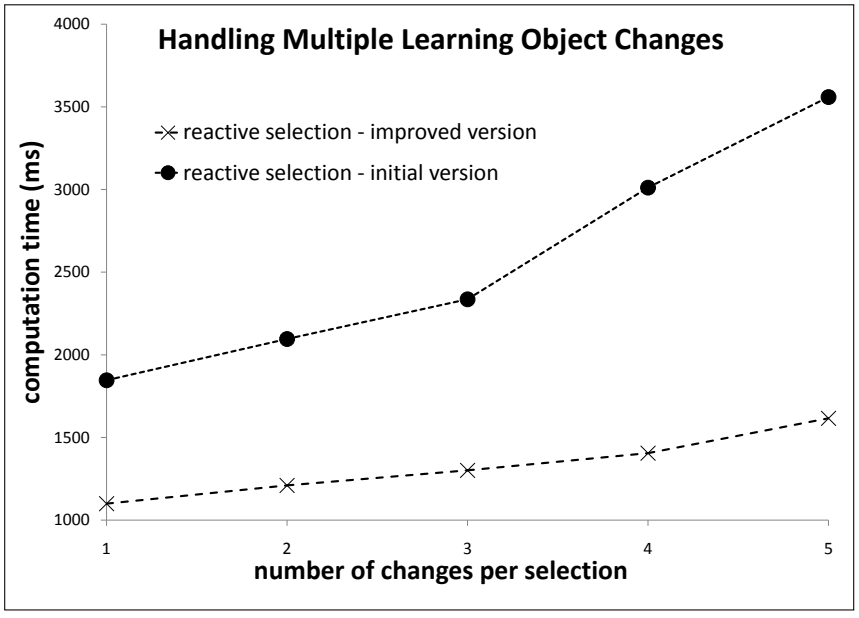

FIGURE 8.19: The effect of the number of changes during selection on selection time

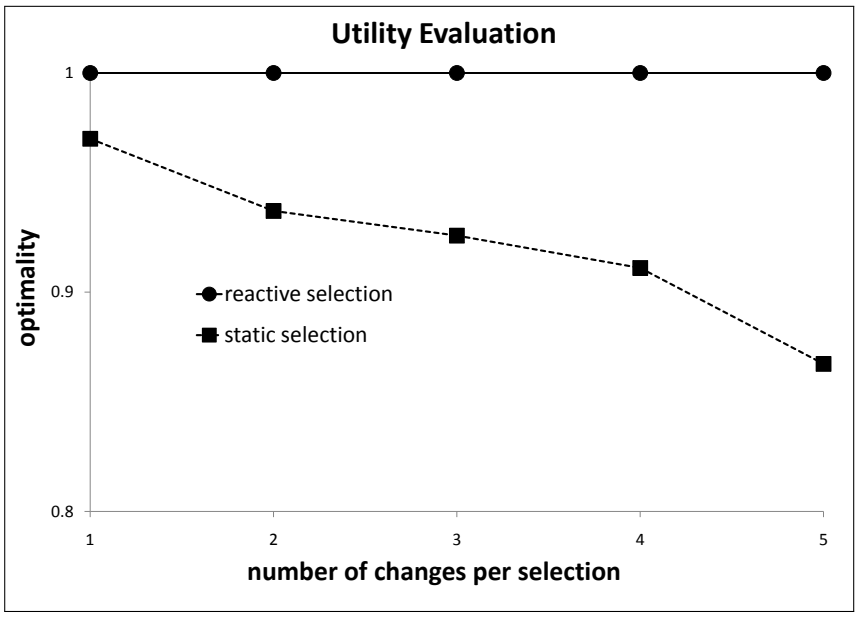

FIGURE 8.20: The gain in optimality achieved by reactive selection

8.6.4 Reactive Execution of Learning Objects

Although the reactive selection algorithm produces the most suitable course for the learner at the time of selection, changes in learning objects can still occur during the delivery (execution) of this course, thus possibly requiring the re-selection of learning objects for the remaining concepts (the concepts not yet presented), in order to guarantee the best learning experience for the user. In what follows, we first show how the proposed reactive execution approach (Chapter 6) should be adapted for the case of learning objects, followed by the evaluation results.

8.6.4.1 Adaptation of the Reactive Execution

Execution-time re-selection should be efficient in order to avoid, as much as possible, any interruption to the execution process (user learning), especially if some changes

are only discovered at a late stage or (in the worst case) at the end of the current component service's execution. Our proposed reactive execution achieves such efficiency through a *light* re-selection approach, in which only a minimal number of modifications to an already existing search graph are needed in response to a change. This approach, however, requires the search graph always to be valid (i.e. to reflect the most recent environment state) prior to change occurrence, and hence involves continuously adjusting the graph in response to all changes during execution. Although effective for short execution periods, such an approach causes an unnecessary overhead (from the composite service provider perspective) if service execution times $exec$ are long $exec >> rslScrtch$ ($rslScrtch$ is the time anticipated to perform the most costly re-selection from scratch, for the non-executed tasks). This is because, when a service execution spans a long period of time $[t_s, t_e]$, it becomes unnecessary to continuously react to changes for this entire duration. Instead, all changes could be ignored until $t_{crt} = t_e - rslScrtch$ (see Figure 8.21), at which point re-selection from scratch for the remaining tasks should be instantiated, in order to restore a valid instance of the search graph with respect to the new environment state. From this point, the execution should continue in the proposed light reactive manner, efficiently accommodating all the changes occurring during the critical interval $[t_{crt}, t_e]$ (the interval signaling the end of the current service execution). This adaptation of the execution algorithm saves the cost of maintaining the search graph and triggering re-selection a potentially very large number of times, during long intervals $[t_s, t_{crt}]$.

Learning objects are examples of services with long execution durations, where the presentation of each learning object usually lasts for at least its typical learning time, and thus the above adaptation to the execution algorithm should be applied here. For instance, even when a learning object runs for several hours, the reactive execution needs to be instantiated only a short time (e.g. 5 minutes) before the end of its execution.

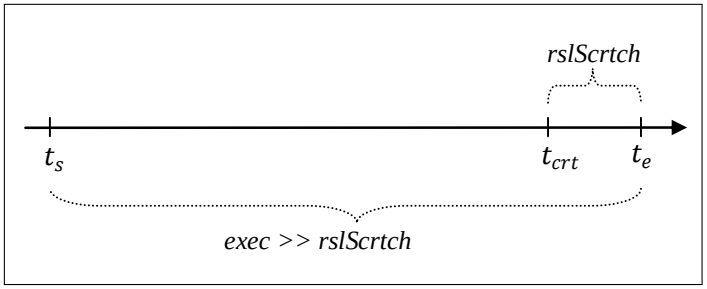

FIGURE 8.21: A Long Execution Interval $[t_s, t_e]$

8.6.4.2 Experimental Results

Like previously, we replicate here the same experiments performed on the randomly-generated dataset (the experiments of Section 6.7), in order to evaluate the effectiveness of our reactive execution approach in the domain of learning objects. Again, all the results reported are averaged over 30 randomly-generated requests (with two global-level constraints and a utility optimisation requirement). The concept of interest is assumed to be *Basic Data Structures* (with up to 11 tasks per abstract plan in the search graph), while the percentage of candidate learning objects per task is set to 100% (i.e. all candidate learning objects are considered for each task). Note that, since only the critical interval $[t_{crt}, t_e]$ of a learning object execution is relevent for evaluating the performance of our reactive execution (during which we risk an innteruption in execution as a result of a changce), we assume in what follows a short execution time per learning object, ignoring the irrelevant long interval $[t_s, t_{crt}]$. This simplifies the experiments and facilitates averaging the results over multiple runs.

Figures 8.22 and 8.23 reproduce the experiments of Figures 6.5(a) and 6.5(b), respectively, which assess the gain in efficiency obtained by reacting to changes through minimal modifications to already existing optimal instances (*reverse* graph approach), as opposed to the re-selection from scratch reaction (*forward* graph approach). The results in Figures 8.24-8.27 correspond to the optimality experiments of Figures 6.6(a)-

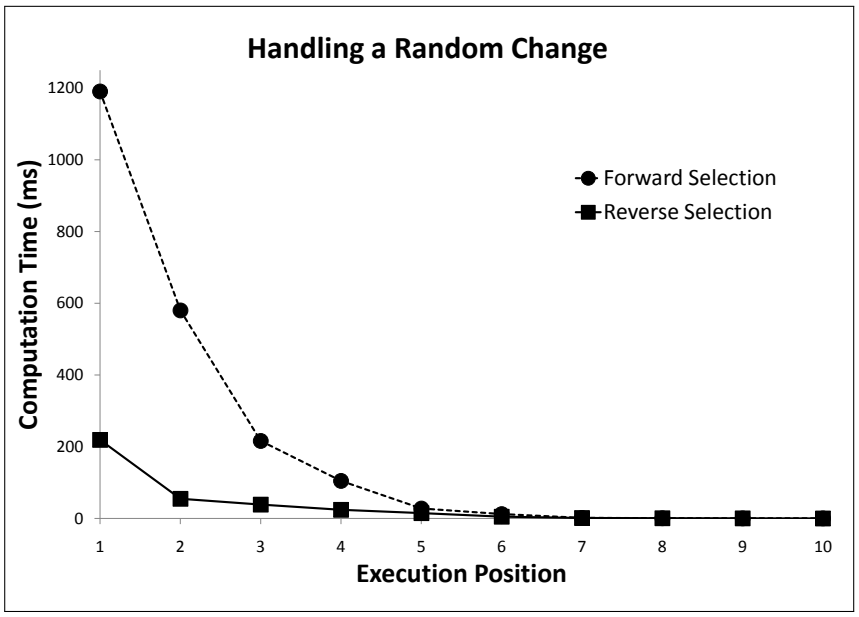

FIGURE 8.22: The gain in efficiency achieved by the reverse graph approach - Random Change

6.6(d), where the early reaction to a change (as soon as it occurs) is compared against the delayed reaction (after invoking the unavailable or violating service), in terms of solution optimality. Finally, the interruption time between service executions is studied in Figures 8.28-8.30, with respect to various factors (similarly to Figures 6.7, 6.8(a) and 6.8(b)). As can be seen, the conclusions derived from the above graphs are similar to those of Section 6.7.

8.6.5 Correlation-aware Selection of Learning Objects

As discussed in Section 8.5.6, learning objects may not be independent from each other regarding quality values. In particular, given the metadata elements selected in this case study, quality correlations can exist among learning objects with respect to three attributes: *typical learning time*, *difficulty*, and *cost*, examples of which are provided in Section 8.5.6. To simulate such correlations, we assume two values per each dependent

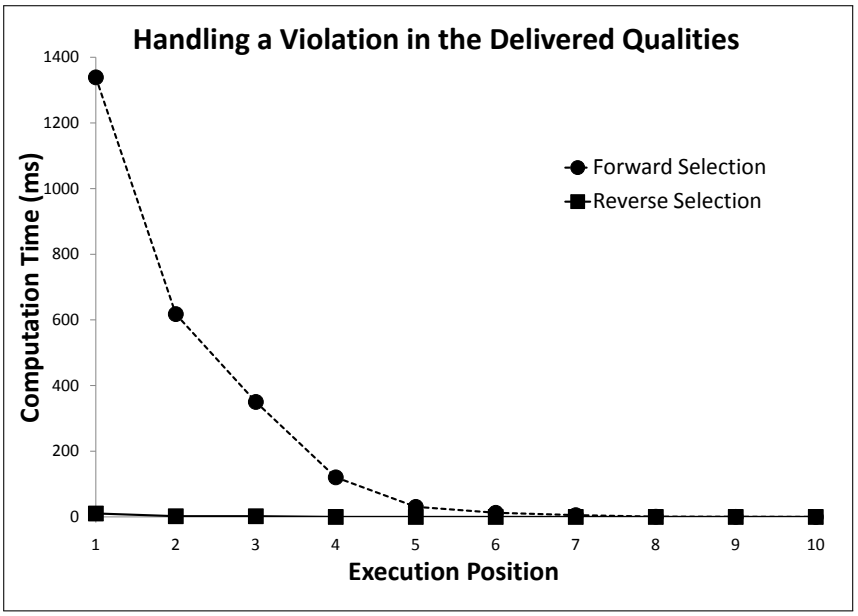

FIGURE 8.23: The gain in efficiency achieved by the reverse graph approach - Violation in Delivered Qualities

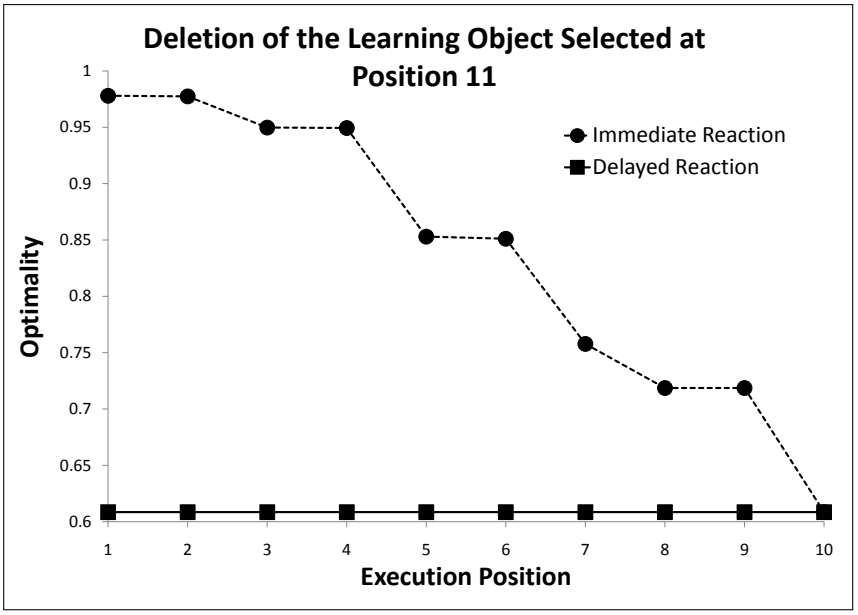

FIGURE 8.24: The effect of execution position on optimality - Deletion Case

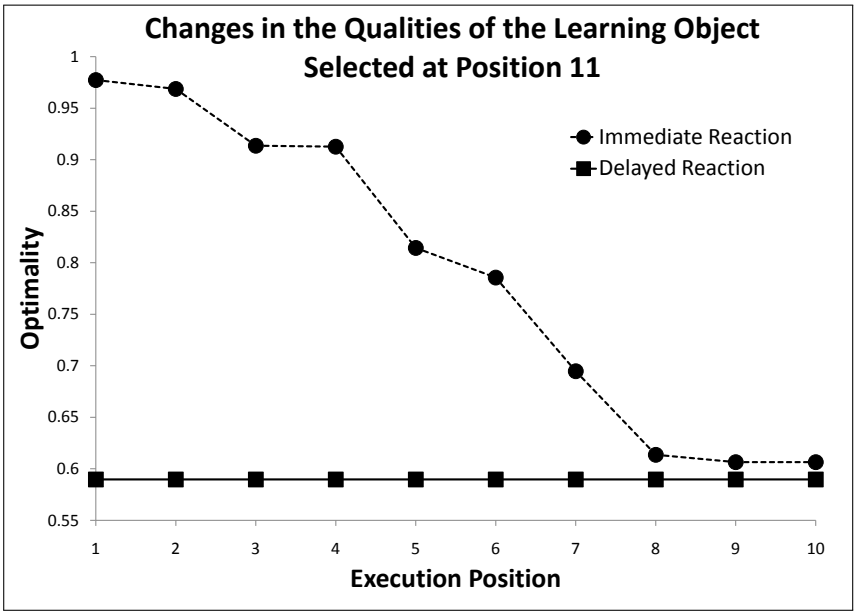

FIGURE 8.25: The effect of execution position on optimality - Quality Changes Case

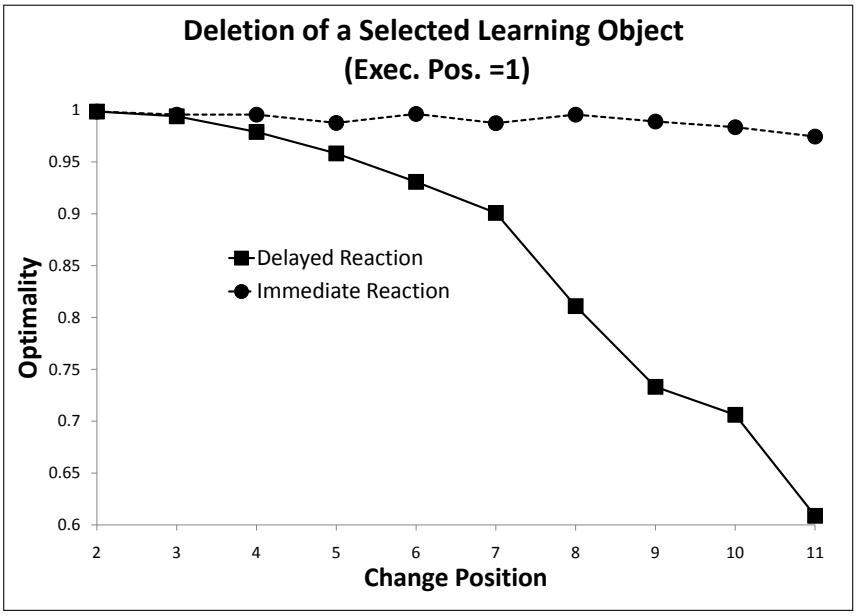

FIGURE 8.26: The effect of change position on optimality - Deletion Case

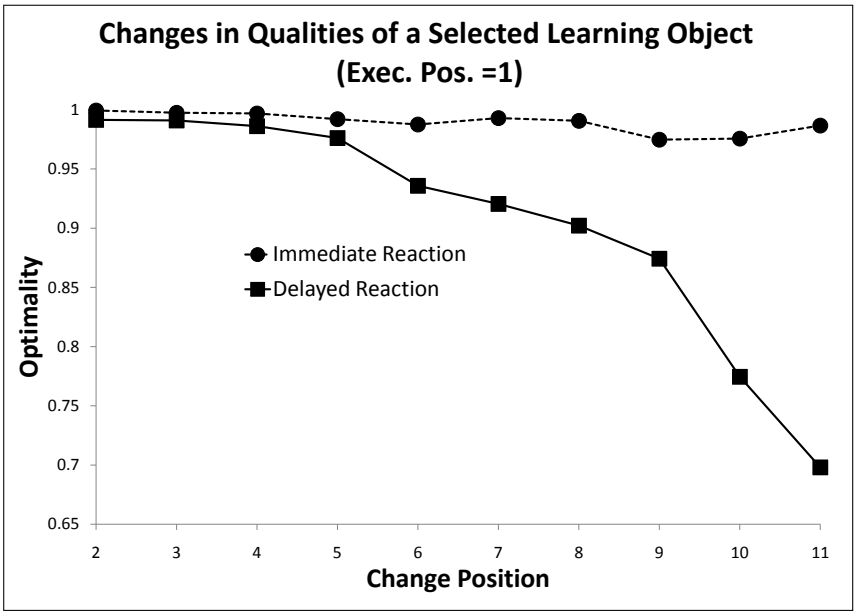

FIGURE 8.27: The effect of change position on optimality - Quality Changes Case

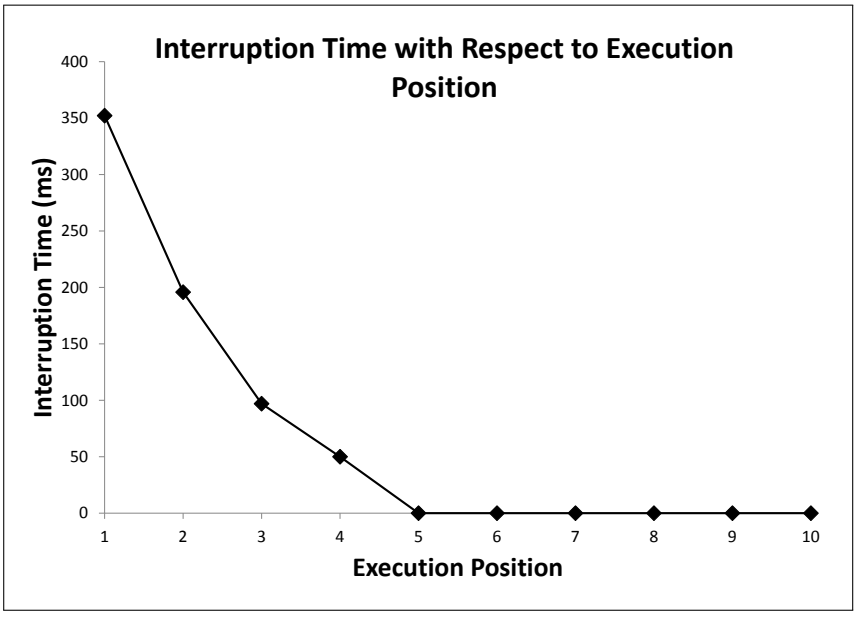

FIGURE 8.28: The effect of execution position on interruption time

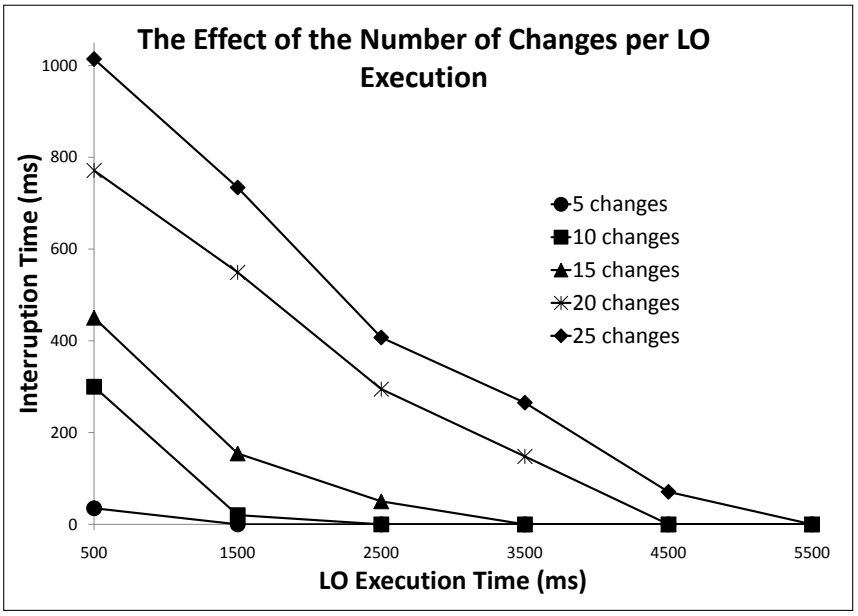

FIGURE 8.29: The effect of the number of changes on interruption time

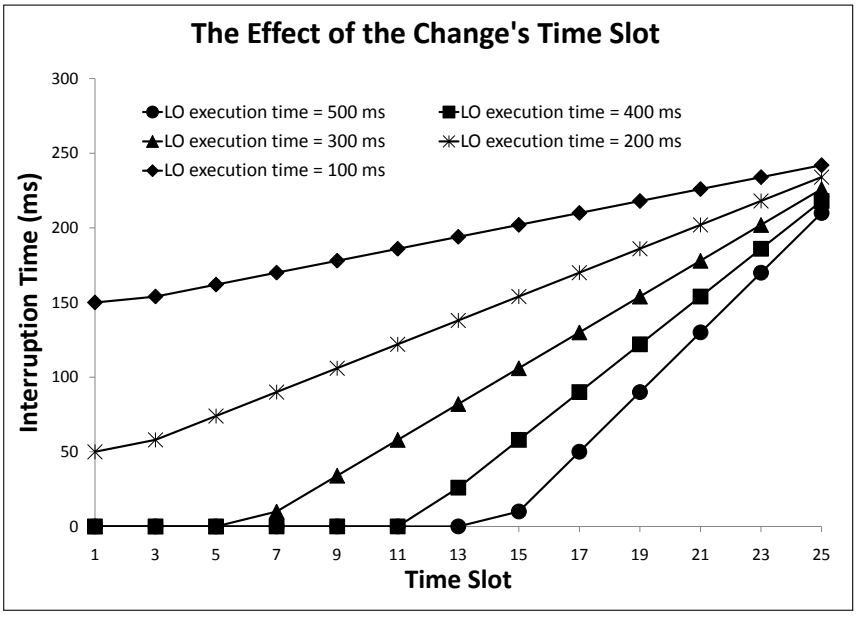

FIGURE 8.30: The effect of change time slot on interruption time

attribute of a learning object: a *default* value, which is provided in the metadata instance of the learning object; and a *correlation* value, which is generated randomly either in set {very low, low, medium, high, very high} (for the *difficulty* attribute), or according to the minimum and maximum estimates of Table B.1 (for the *typical learning time* and *cost* attributes). The learning context condition ctx associated with the correlation value is generated randomly, while the one of the default value is set to $\neg^{dnf}(ctx)$.

Again, we repeat the experiments previously performed on the randomly-generated dataset (the experiments of Section 7.7), in order to evaluate the effectives of our correlation-aware selection algorithm on the space of correlated learning objects. Specifically, Figure 8.31 shows the reduction in computation time achieved by our correlation-aware pruning, while Figures 8.32-8.34 study the effect of various factors on the complexity of our algorithm, namely the effect of correlation percentage, number of correlated tasks, complexity of context conditions, and number of dependent attributes. Finally, Figures 8.35 and 8.36 illustrate the improvement in optimality archived by correlation awareness, in the presence of positive and negative correlations, respectively. Conclusions similar to those of Section 7.7 can be drawn from the above figures.

Note that all the results reported are averaged over 30 requests concerning the *Basic Data Structures* goal concept, with a maximum of 11 tasks per abstract plan. Request quality constraints are generated randomly (each request is assumed to have two global constraints), while request quality weights are set to 0.166 for all the global attributes (i.e. all the global attributes are equally important). Other experimental settings are provided in Table 8.6.

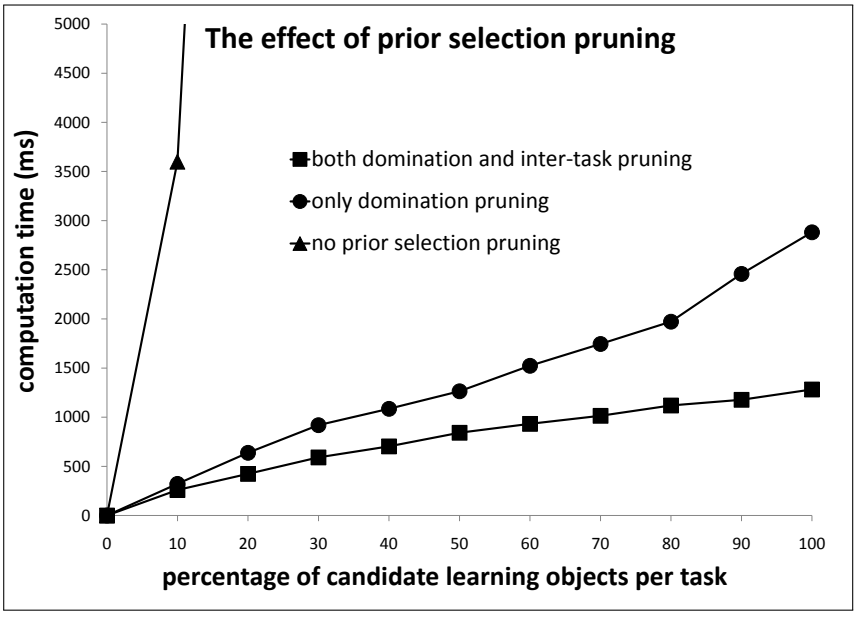

FIGURE 8.31: The effect of correlation-aware pruning on selection time

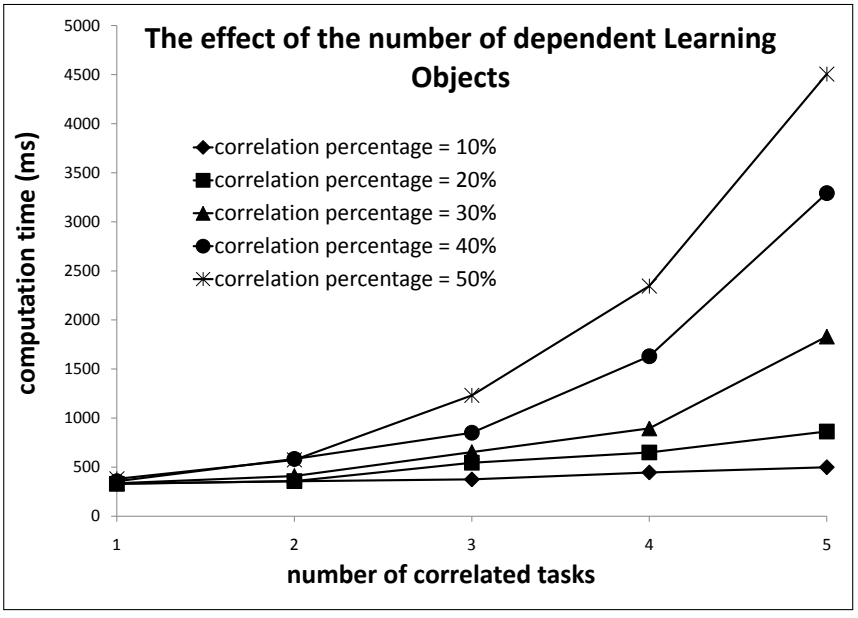

FIGURE 8.32: The effect of the number of dependent learning objects on selection time

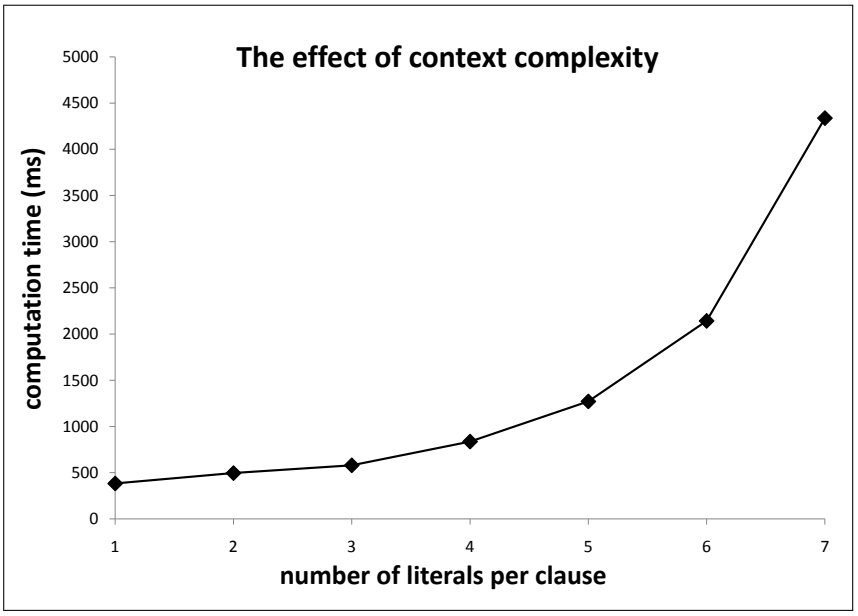

FIGURE 8.33: The effect of context complexity on selection time

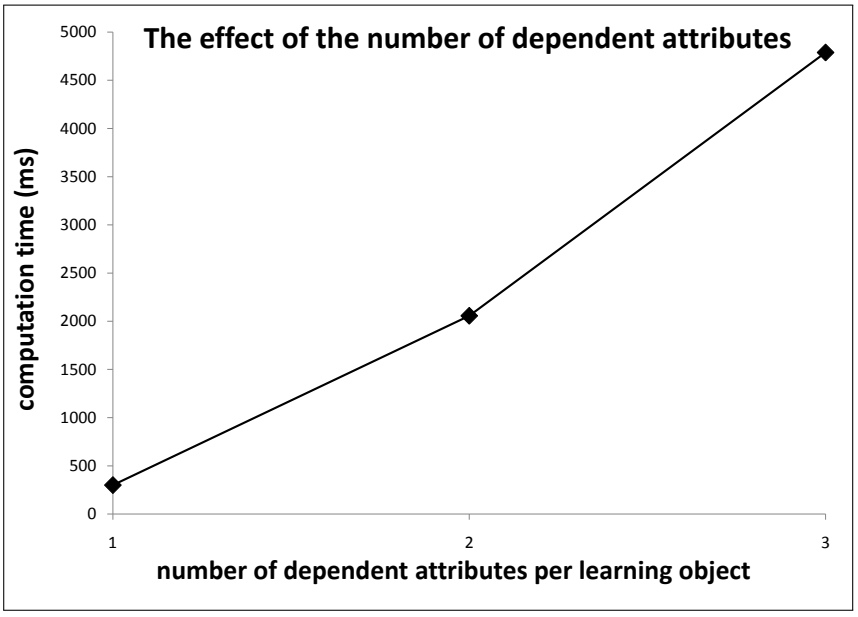

FIGURE 8.34: The effect of the number of dependent attributes on selection time

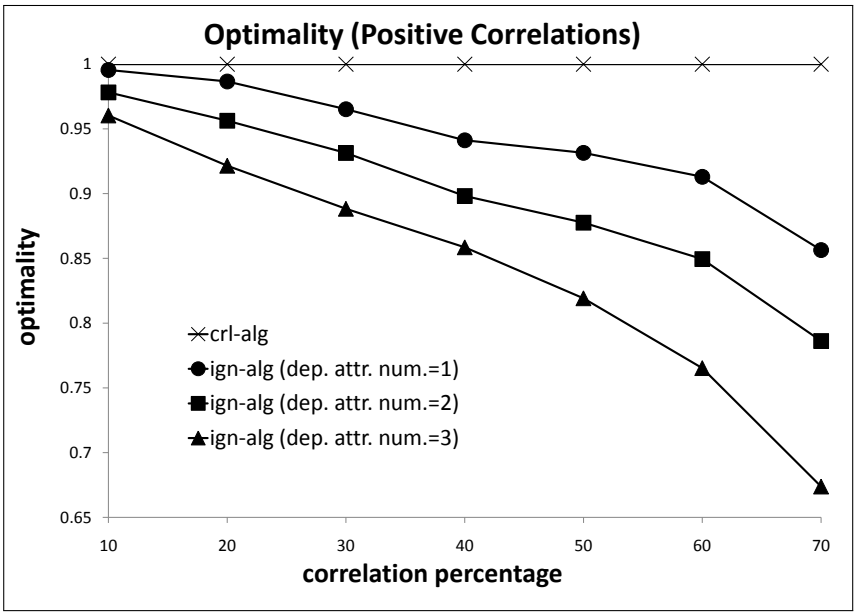

FIGURE 8.35: The gain in optimality achieved by correlation-aware selection (positive correlations)

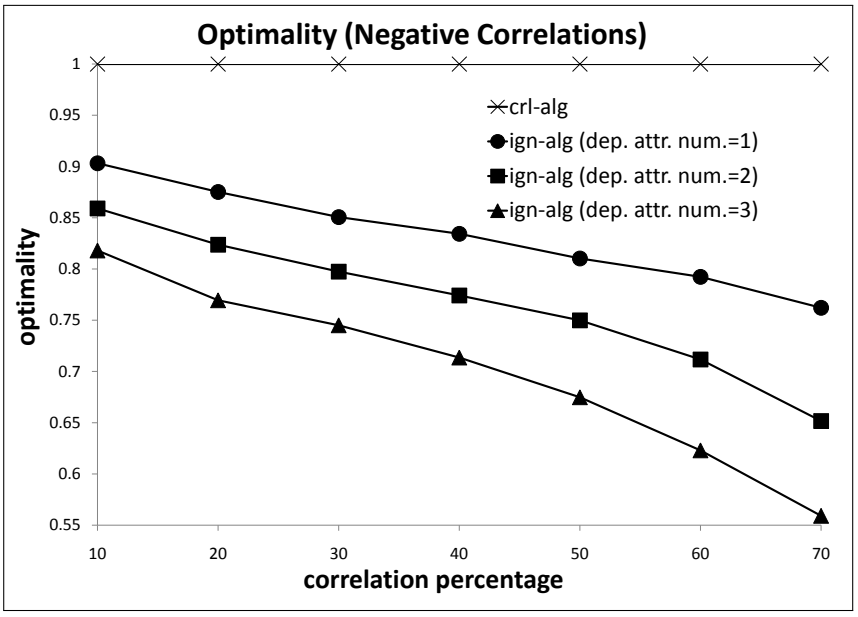

FIGURE 8.36: The gain in optimality achieved by correlation-aware selection (negative correlations)

TABLE 8.6: Experimental Settings

	Figure 8.31	Figure 8.32	Figure 8.33	Figure 8.34	Figure 8.35	Figure 8.36
Percentage of candidate LOs per task	between 10% and 100%	100%	100%	100%	100%	100%
Correlation percentage	20%	between 10% and 50%	10%	10%	between 10% and 70%	between 10% and 70%
Number of dependent attributes	1	1	1	between 1 and 3	between 1 and 3	between 1 and 3
Number of correlated tasks	4	between 1 and 5	1	1	4	4

8.7 Conclusion

In this chapter, we have demonstrated the practical value of the techniques and algorithms developed in this thesis through a real world example, concerning the dynamic composition of learning objects (reusable learning materials) in the e-learning domain. Learning objects form a suitable framework for the purpose of our evaluation, due to several reasons. Being exposed through uniform, machine-readable descriptions (IEEE LOM metadata), and published for discovery and reuse through public repositories, these objects meet the definition of services. Moreover, learning objects are heterogeneous in their granularities (they can range from a single image to a whole module), which makes them particularly attractive for evaluating our multi-granularity planning structure. Since hundreds of learning objects can be available for each concept with varying properties (e.g. interactivity level, difficulty level, duration, etc.), these objects represent a good example of a large and diverse service space, where applying pruning techniques during composition is essential to reduce complexity. Furthermore, large numbers of new learning objects are made available every day, while existing learning objects can be updated or become unavailable, at any time. Due to the long running nature of learning objects, such changes in the learning object repository are especially

likely to occur during the delivery (execution) of a selected course. This demands effective change handling mechanisms to guarantee a suitable and interruption-free learning experience for the learner (user). Finally, with the potential dependencies among learning objects in relation to many features, correlation-awareness is important to provide accurate estimations of course properties.

A thorough experimental evaluation has been conducted for all our algorithms, in static, dynamic, and correlated environments. The results obtained validate our previous findings, achieving scalability and optimality in all settings.

Chapter 9

Conclusions and Future Work

Open distributed service-based systems are becoming increasingly common, in which providers encapsulate their offerings, ranging from expensive hardware components to entire applications, within services, and expose them over open networks of customers. These services can then be automatically discovered, invoked, and composed over the network using agreed standards and protocols, thus allowing a rapid and low-cost development of complex distributed applications spanning organisational boundaries. In such systems, a large number of providers could be competing for the same type of service, but at varying quality offerings. Even at a particular service level, several quality variations may be possible, depending on the usage context of this service. Furthermore, due to the distribution, participant autonomy and lack of local control, such systems operate in highly dynamic and uncertain environments, in which services can be added, removed or change their characteristics, at any time. Within these settings, selecting the best combination of services for the user, and ensuring that the selected solution remains available, satisfactory, and optimal during the different composition steps is not trivial, and despite active effort, still has not been addressed adequately. In response, this thesis has made a number of novel contributions towards addressing

these challenges, which are summarised in Section 9.1, followed by a list of possible future extensions of our work in Section 9.2.

9.1 Research Summary

In what follows, we summarise our contributions with respect to the research challenges stated at the beginning of this thesis (Section 1.3).

9.1.1 Granular Heterogeneity

To address the granular diversity of services in open service-based systems, we have proposed in Chapter 3 a multi-granularity representation of composition planning knowledge. The various granularity levels of tasks allow the discovery of services at a vast range of sizes, including special-offer packages and possibly even other composite service providing agents. Guiding selection and facilitating the expression of complex control constructs, this representation maintains the benefits of a pre-determined workflow structure (usually adopted), but it complements these with the ability to explore a large variety of planning alternatives. This rich collection of plans is combined into a search graph, the plan paths graph, in Chapter 4, to be utilised during the selection and execution processes.

A practical example of such planning knowledge representation has been provided in Chapter 8, where it serves as a domain structure for composing learning objects, while accommodating the diverse granularity levels at which these objects are usually available.

9.1.2 Time Constraints

Achieving time efficiency has been a main concern in designing the techniques and algorithms throughout the thesis. In particular, in Chapter 4, search space pruning techniques have been proposed to reduce the number of service combinations that need to be considered during service selection. The user request at hand (in terms of the quality constraints and quality optimisation requirements) is the primary basis for such pruning: plans and services that are not promising in the context of this request are eliminated prior to selection. As opposed to the heuristic proposals in the literature, our pruning is optimality-retaining. It is also algorithm-independent, i.e. it can be applied as pre-processing to any selection algorithm. The pruning techniques were generalised in Chapter 7, to accommodate the more challenging case where the search space experiences inter-service quality dependencies. This generalisation preserves the ability to achieve a significant reduction of the search space, at no optimality loss. To further increase efficiency, similar pruning is conducted at the (sub-)composite service level, while the selection algorithm is progressing (in both the correlation-ignorant and correlation-aware spaces).

When designing the reactive version of the selection algorithm (Chapter 5), rules were derived to reason effectively about the need to react to a change encountered during selection, and to achieve such a reaction (if needed) in a timely manner, through efficiently repairing the affected part of the search graph, avoiding expensive recalculations from scratch.

At the execution stage, time inefficiency could be measured in terms of the interruption time that the user is subjected to as a result of unexpected situations. In this regard, we have proposed in Chapter 6, the combination of the following three factors in order to reduce such inefficiency.

Parallel-to-execution reaction. The adaptation process is instantiated at the earliest possibility during a component execution, so that the chance of completing the adaptation before this execution terminates is maximised, and thus execution disruptions are minimised.

Light reselection. In order to keep the efficient repair rules of reactive selection relevant for execution-time reselection, a reverse version of the search graph is assumed, thus facilitating a fast reaction in response to any change, especially in the critical case where the change concerns the task being invoked.

Change prioritisation. Each change encountered during execution goes through an assessment process to derive its priority regarding the situation at hand. Specifically, changes potentially affecting action points in the near future are handled urgently, while the adaptation to those of less importance is allowed to be carried out during the next component execution, without causing interruption. Through such a novel and comprehensive analysis of changes, and the corresponding behaviour of the executing system, inefficiencies are avoided, unless necessary.

9.1.3 Dynamism and Uncertainty

The dynamism and uncertainty inherent in service-based systems are demanding that composite applications be equipped with adaptation capabilities, in order to deal with such volatility. Ideally, a run-time composition system should possess the ability to (**G1**) recover from unexpected situations on their occurrence, (**G2**) exploit new emerging opportunities to enhance the selected solution at any selection or execution stage, (**G3**) proactively prevent future breaches and faulty behaviour, by performing early corrective actions, while appropriate alternatives are still available, (**G4**) produce an optimal solution by any instantiated service (re-)selection process, and (**G5**) maintain

efficiency at all times, especially keeping triggered adaptations transparent to the end user.

Current state-of-the-art approaches to adaptation usually achieve some of the above goals at the expense of others (see Table 9.1 for a comparison of the approaches in terms of these goals). We believe that we are the first to propose an adaptive composition approach combining *all* of these features, as summarised below.

G1: Recovery on service violation or unavailability. In situations where it is not possible to prevent undesired behaviour prior to its execution, our approach allows very fast recovery with minimal interruption, yet effectively reasoning about the best forward replacement available.

G2: Seizing emerging opportunities. Whenever a chance for improving the current selection is identified as a result of a change in the environment, adaptation is triggered to exploit such an opportunity.

G3: Violation prevention. This is achieved at two levels, the selection level and the execution level. By incorporating reaction capabilities into the selection process (which we are the first to propose), any service unavailability or quality violation arising at the time of selection, is handled at that point, thus ensuring that such problematic services are not selected for execution. Furthermore, as execution advances, problems encountered in component services scheduled for future execution are dealt with as early as possible, through appropriate re-assignment of services to tasks, before reaching erroneous execution points where recovery opportunities would be of lower quality and more costly.

G4: Selection Optimality. The produced combination of services for the non-executed tasks, by any triggered re-selection process, is always the best possible given the tasks already executed (if any), and the current state of services in the environment.

TABLE 9.1: Comparison of adaptive composition approaches (see Table 2.3 for further details)

	G1	**G2**	**G3**	**G4**	**G5**
Our approach	✓	✓	✓	✓	✓
Current proactive attempts	✓	✗	✓	✗	partially
Reaction on violation:					
execution-time heuristic re-selection	✓	✗	✗	✗	partially
execution-time optimal re-selection	✓	✗	✗	✓	✗
pre-computed backups	partially	✗	✗	✗	✓

G5: Adaptation transparency. Through continuous, parallel-to-execution, and light reaction to changes, the search graph is kept up-to-date with the most recent environment state. This facilitates very quick repair to the search graph when an affecting service change occurs at a critical period, i.e. at a late stage or at the end of a component execution, causing almost no interruption, which is demonstrated via our experimental results.

9.1.4 Service Correlations

Despite their significant impact on the quality of the solution delivered to the end user, quality correlations among services have largely been ignored by current service composition approaches. To address this, we have presented a novel correlation-aware selection approach in Chapter 7. Specifically, service quality offerings in our basic service model are generalised to become multi-valued and context-enriched, thus allowing expression of a variety of inter-service quality dependencies. Through a service space transformation, replacing each multi-valued (per quality attribute) service with a number of single-valued *constrained* representatives, the ability to reason effectively about service suitability for the current request and to consequently achieve considerable pruning, both prior to and during service selection, is maintained in the presence

of quality correlations. With respect to this, suitable search space reduction techniques and an efficient selection algorithm, extending the original correlation-ignorant proposals, are suggested.

9.2 Future Directions

In what follows, we list a number of future directions that we believe are interesting extensions and improvements to our work.

9.2.1 Planning Knowledge Modification

While the allocation of services to tasks in our approach is performed dynamically at run time, the compositional planning knowledge is assumed to be fixed at design time (e.g. generated by a human expert). A fully adaptive system, however, should be capable of modifying its planning structure on the fly, if such modification is deemed necessary. In the context of the open, inter-organisational settings we consider, this is especially important because of the diversity and the highly dynamic nature of the participating components. In particular, the dynamics of the environment could cause previously valid plans to become partially not executable due to the disappearance of some types of services. Similarly, emerging services at new granularity levels could open up alternative (better) possibilities for planning complex goals unforeseen at design time. To this end, it is possible to investigate different means for dynamic learning (adaptation) of the planning knowledge hierarchy in response to environment changes, which include learning by *exploration* (composing and decomposing existing tasks to discover new opportunities in the service space); learning by *observation* (acquiring new sequences of actions from both monitoring the execution traces of relevant specialised agents [135], and analysing the inner structure (if available) of candidate compound

services); and learning by *interaction* (exchanging plans with other composite service provider agents, i.e. through explicit consultation requests).

9.2.2 Reactive Execution with Service Rollback

In our work (specifically in Chapter 6), we have concentrated on a *forward* recovery mechanism to accommodate the changes encountered in the environment during execution. In other words, only the tasks that are not yet executed can be re-assigned to other services or replaced with other tasks from an alternative plan, while those already executed are considered final with their assignments being unchangeable. Although this is a plausible assumption for some types of services (examples of which are the learning objects evaluated in the case study, where it does not seem reasonable to make the user repeat the learning process for an already acquired concept), it may be restrictive for other types whose effects could be undone (for instance, it is usually possible to cancel an already-booked hotel reservation, even if this incurs some penalty). Hence, in the latter case, a more powerful adaptive system could be achieved by extending our recovery mechanism to allow for the possibility of rolling back the composite service execution to a previous point in time. This could be either to re-execute some tasks with more promising services, or to switch to an alternative course of action facilitating a successful completion of the goal at hand, when reaching a deadlock with the currently available forward options. Such *backward* recovery requires a richer service model reflecting the compensation capabilities of services [28, 55] with the associated fee policies, and an extended reasoning process quantifying and incorporating the cancellation consequences when making adaptation decisions.

9.2.3 Trust-aware Service Model

The approach proposed in this thesis is tailored towards dynamic and uncertain environments, where service providers are possibly autonomous and self-interested, and their quality offerings are not necessarily trustworthy. However, the current version does not provide any support for reasoning about the degree of reliability of these offerings when performing service selection. That is, all services are assumed to have the same credibility regarding their promised quality values. In the absence of previous interactions with service providers (e.g. this might be the case in highly dynamic provider populations), it may be difficult to distinguish one service provider from another with respect to the reliability of their quality offerings. Yet, when available, such experience information could provide a useful guide for prioritising interaction partners when making selection decisions. Many trust and reputation models based on this information have already been developed in the area of agents and multi-agent systems to assess the trustworthiness of a particular party [106, 119]. It would thus be interesting to investigate how such models could be adopted in the context of our work in order to improve the service selection process. In other words, rather than purely relying on *adaptive* actions, some kind of *preventive* behaviour could be accomplished through augmenting the quality advertisements of services with trust or reliability scores, and then utilising these scores to select more honest and reliable services, thus reducing the chance of deviations at execution time.

9.2.4 Consideration of Neglected Implementation Costs

Time constraints and resource boundedness of real-world systems (where change monitoring, composition (re-)planning, and composition execution would possibly be competing for the same system resources) may impose additional complications (costs)

currently unaccounted for (assumed to be free) in this thesis. In particular, the following require further consideration.

- *Change Monitoring Overhead.* In this thesis, we assume the ability to *continuously* observe the environment state in order to detect changes when they occur. Such regular monitoring, however, could consume significant computational resources (e.g. due to repeated costly actions related to service registry access or historical-data-based performance prediction), and hence may not be practical, especially in cases where the environment is characterised by low dynamism.

  Moreover, changes are assumed to be detected *instantly* on each environment check (i.e. the non-dominance sets associated with tasks are assumed to be up-to-date at each moment), ignoring the delay possibly associated with the detection process as a result of expensive calculations or other reasons. For example, a service may provide no notification when experiencing failure. Rather, it may crash silently (either intentionally or unintentionally), and thus could only be marked as failed/unavailable after an appropriate time-out period has elapsed. It is worth noting that managing failure-related delays is particularly important during execution, where a long waiting time for a response after invoking a faulty service could jeopardise the feasibility of the entire solution.

- *Reactive Planning/Re-planning Overhead.* In this thesis, we assume that the frequency of changes in the environment does not affect the *termination* of the proposed reactive algorithms, i.e. the dynamism degree during any running selection or re-selection process only necessitates a finite number of roll-backs. Although this is a realistic assumption in many cases, some specific settings with very high change rates require further time and resource awareness. This is because, an environment exhibiting continuous volatility could result in infinite re-planning

to accommodate such changes, thus exhausting system resources and preventing execution progress (i.e. postponing execution indefinitely).

With respect to the above, a trade-off between the different steps involved in the proposed adaptive composition could be investigated to maximise overall system performance. For example, instead of only focusing on optimising the quality features of the execution-time solution, the time allocated to change detection, initial solution selection, and solution re-selection could also be incorporated as affecting criteria into the optimisation process.

9.2.5 Other Immediate Extensions

Immediate possibilities for extending the work in this thesis includes the following.

Heuristic-based Pruning. The pruning techniques proposed in this thesis are optimality-directed. That is, they only eliminate the combinations guaranteed not to be of interest, while preserving all the candidates for an optimal solution. Despite generally proving to achieve efficiency in selection, such pruning is largely dependent on the quality distributions of services and the imposed request, and hence may not scale well in some specific situations (e.g., particular quality settings where no service dominates another due to negatively related attributes). Therefore, it would be valuable to investigate suitable heuristic-based alternatives for such situations, which could compromise optimality to some degree in favour of pruning more combinations, so that selection scalability is ensured.

Complex Control Constructs. Although it is possible to express complex planning structures through our composition knowledge representation, our selection algorithm, in its current version, is limited to the sequential structure. This algorithm could be extended to account for other types of structure, especially parallel and loop constructs.

Correlation Context Reduction. Our correlation-aware selection approach relies heavily on manipulating the context conditions associated with services during the different stages. The complexity of these conditions thus plays a major role in increasing the complexity of the selection process (as shown by our analysis). To tackle this, there are different possibilities for decreasing the size of context conditions, including applying join operations on their comprising clauses, and introducing ways to eliminate redundancies in these clauses that are more effective than the simple inclusion check currently adopted.

Reactive Selection and Execution under Correlations. The solution presented in this thesis for handling inter-service quality correlations does not account for environment dynamism during selection and execution. Designing appropriate extensions of our reactive algorithms, capable of operating in correlation-aware service environments, is thus a target area for future research. It is worth noting that with the presence of QoS correlations between services, managing changes to services becomes more challenging, especially in the cases where the dependent services precede the dependee services in execution order. This is because, in such cases, any unavailability on the dependee side after executing the dependent, will not only require re-selection to recover from the situation, but also adequate ways to handle the violation in the composition context of the executed (dependent) service.

Appendix A

Optimality Proof for Correlation-aware Pruning

In what follows we prove that, in our correlation-aware selection approach, applying the proposed pre-selection context restriction techniques to reduce the search space does not affect the ability to find the optimal solution.

A.1 Pre-domination Context Adjustment

Adjusting the context conditions of the services in S^c prior to domination pruning, i.e. according to Equation 7.1, does not eliminate any *valid* composition from the search space, where valid here refers to being possible from the perspective of all the component services. The proof is straightforward. For any service $s \in S^c$, adjusting $ctx(s)$ by Equation 7.1 removes from the search space all the compositions containing service s (i.e. satisfying $ctx(s)$), but not satisfying condition $\neg(sel(task(s')) \in \{s'\}) \vee ctx(s')$. That is, all the compositions removed satisfy condition $(sel(task(s')) \in \{s'\}) \wedge \neg ctx(s')$.

These compositions are not possible from the perspective of service s', and hence are not valid.

Note that, since $ctx_d(s)$ represents a more strict version of $ctx(s)$, the following holds for any composite service cs in S^c:

$$satisfies(cs, ctx_d(s)) \Rightarrow satisfies(cs, ctx(s)) \tag{A.1}$$

where Function $satisfies(cs, c^\vee)$ returns true if composite service cs satisfies condition $c^\vee$, i.e. $cs \cap^{dnf} c^\vee \neq \emptyset$ (cs is considered a conjunction of positive propositional variables whose ranges correspond to cs's component services).

A.2 Domination-based Pruning

Consider two services $s_i, s_j \in candidates^c(t)$, such that s_i **r-dm** s_j, with their respective pre-pruning context conditions being $ctx_d(s_i)$ and $ctx_d(s_j)$. As a result of domination-based pruning, $ctx_d(s_j)$ is restricted to $ctx_d(s_j) \setminus^{dnf} ctx_d(s_i)$. Such restriction eliminates from the search space all the compositions containing service s_j and satisfying condition $ctx_d(s_j) \cap^{dnf} ctx_d(s_i)$. Thus, we need to prove that none of the eliminated compositions is a candidate for an optimal solution. That is:

$$\forall cs \in comp^c(task_r) \text{ such that } satisfies(cs, addltr(ctx_d(s_j) \cap^{dnf} ctx_d(s_i), s_j)), \\ cs \text{ is not a candidate for an optimal solution} \tag{A.2}$$

where $comp^c(task_r)$ is the set of all *valid* compositions in the correlation-aware candidate space S^c.

We provide a proof by contradiction. Assume Equation A.2 does not hold, i.e.

$$\exists cs_j \in comp^c(task_r),$$
$$satisfies(cs_j, addltr(ctx_d(s_j) \cap^{dnf} ctx_d(s_i), s_j)) \ \wedge \ cs_j \text{ is non-dominated} \quad (A.3)$$

Since s_i **r-dm** s_j, composition $cs_i = replace(cs_j, s_j, s_i)$, replacing service s_j with service s_i in composite service cs_j, is not a valid composition; otherwise cs_j would be request-based dominated by cs_i, which contradicts our assumption in Equation A.3. This implies that there exists a component service $s_k \in services(cs_i)$ such that $ctx(s_k)$ is not satisfied by composition cs_i:

$$\exists s_k \in services(cs_i), \ \forall c_k^{\wedge} \in cls(ctx(s_k)), \ \ \neg satisfies(cs_i, c_k^{\wedge}) \quad (A.4)$$

Note that $s_k \neq s_i$, i.e. $s_k \in services(cs_i) \setminus \{s_i\}$. This is because, from our assumption in Equation A.3, we have:

$$\begin{aligned}
& satisfies(cs_j, addltr(ctx_d(s_j) \cap^{dnf} ctx_d(s_i), s_j)) \\
\Rightarrow \ & \forall s \in candidates^c(t), \quad satisfies(replace(cs_j, s_j, s), ctx_d(s_j) \cap^{dnf} ctx_d(s_i)) \\
\Rightarrow \ & satisfies(cs_i, ctx_d(s_j) \cap^{dnf} ctx_d(s_i)) \\
\Rightarrow \ & satisfies(cs_i, ctx_d(s_i)) \quad (A.5) \\
\Rightarrow \ & satisfies(cs_i, ctx(s_i)) \quad \text{(according to Equation A.1)}
\end{aligned}$$

Now, since we assumed cs_j is a valid composition, the following holds:

$$\forall s \in services(cs_j), \ \exists c^{\wedge} \in cls(ctx(s)), \ \ satisfies(cs_j, c^{\wedge}) \quad (A.6)$$

From Equations A.4 and A.6, and from the fact that $s_k \neq s_i$ in Equation A.4, we conclude:

$$\exists s_k \in S^c \setminus candidates^c(t),\ \exists c_k^{\wedge} \in cls(ctx(s_k)),\quad satisfies(cs_j, c_k^{\wedge}) \wedge \neg satisfies(cs_i, c_k^{\wedge})$$

Since $cs_i = replace(cs_j, s_j, s_i)$, $\quad satisfies(cs_j, c_k^{\wedge}) \wedge \neg satisfies(cs_i, c_k^{\wedge})$ implies that:

$$\begin{aligned} &\exists l \in ltr(c_k^{\wedge}),\quad (ltask(l) = t)\ \wedge\ ((sel(t) \in \{s_i\}) \cap^l l = \emptyset)\ \wedge\ ((sel(t) \in \{s_j\}) \cap^l l \neq \emptyset) \\ &\Rightarrow (c_k^{\wedge} \in clst(ctx(s_k), t))\ \wedge\ (c_k^{\wedge} \cap^{cls} (sel(t) \in \{s_i\}) = \emptyset) \end{aligned} \tag{A.7}$$

From Equation A.7, service $s_k \in affctx(s_i)$, and Case 2 is the case applicable for the corresponding pre-domination adjustment of service s_i's context by $ctx(s_k)$. Hence, prior to domination-based pruning, $ctx(s_i)$ was restricted according to Equation 7.1, as follows:

$$\begin{aligned} &ctx_d(s_i) = delltr(c_{ik}^{\vee}, s_i) \\ &\qquad \text{where}\ \ c_{ik}^{\vee} = addltr(ctx(s_i), s_i) \cap^{dnf} (\neg(sel(task(s_k)) \in \{s_k\}) \vee ctx(s_k)) \end{aligned} \tag{A.8}$$

By the definition of operator $\cap^{dnf}$, the conjunctive clauses of condition $c_{ik}^{\vee}$ in Equation A.8, are given as follows:

$$\begin{aligned} cls(c_{ik}^{\vee}) = \{c_{ik}^{\wedge} \neq \emptyset \mid\ &\exists c_i^{\wedge} \in cls(addltr(ctx(s_i), s_i)),\ \exists c_k^{\wedge} \in cls(ctx(s_k)), \\ &(c_{ik}^{\wedge} = c_i^{\wedge} \cap^{cls} c_k^{\wedge})\quad \vee\quad (c_{ik}^{\wedge} = c_i^{\wedge} \cap^{cls} \neg(sel(task(s_k)) \in \{s_k\})) \end{aligned} \tag{A.9}$$

Note here that $\forall c_{ik}^{\wedge} \in cls(c_{ik}^{\vee}),\quad (sel(t) \in \{s_i\}) \in ltr(c_{ik}^{\wedge})$, which follows from the definitions of Function *addltr* and Operators $\cap^{dnf}$, $\cap^{cls}$, and $\cap^{l}$.

Let's now analyse the two possibilities for $c_{ik}^{\wedge}$ in Equation A.9. When $c_{ik}^{\wedge} = c_i^{\wedge} \cap^{cls} c_k^{\wedge}$, and since $\neg satisfies(cs_i, c_k^{\wedge})$ (from Equation A.4), it follows that $\neg satisfies(cs_i, c_{ik}^{\wedge})$.

Similarly, when $c^{\wedge}_{ik} = c^{\wedge}_{i} \cap^{cls} \neg(sel(task(s_k)) \in \{s_k\})$, and since s_k is a component service of cs_i, we can easily conclude that $\neg satisfies(cs_i, \neg(sel(task(s_k)) \in \{s_k\}))$, leading to $\neg satisfies(cs_i, c^{\wedge}_{ik})$. From the above two cases, we have:

$$
\begin{aligned}
& \forall c^{\wedge}_{ik} \in cls(c^{\vee}_{ik}), \quad \neg satisfies(cs_i, c^{\wedge}_{ik}) \\
\Rightarrow \quad & \neg satisfies(cs_i, c^{\vee}_{ik}) \\
\Rightarrow \quad & \neg satisfies(cs_i, delltr(c^{\vee}_{ik}, s_i)) \quad \text{(note that } s_i \in services(cs_i)\text{, and thus removal} \\
& \text{of constraint } (sel(t) \in \{s_i\}) \text{ from } c^{\vee}_{ik} \text{ does not affect the satisfaction status of } cs_i) \\
\Rightarrow \quad & \neg satisfies(cs_i, ctx_d(s_i)) \qquad \text{(A.10)}
\end{aligned}
$$

The conclusion in Equation A.10 contradicts, however, the assumption of Equation A.3 that $satisfies(cs_j, addltr(ctx_d(s_j) \cap^{dnf} ctx_d(s_i), s_j))$, leading to $satisfies(cs_i, ctx_d(s_i))$ (see the derivation of Equation A.5). Hence, this assumption cannot be true, which proves Equation A.2. In other words, any valid non-dominated composition containing the dominated service s_j, cannot satisfy $ctx_d(s_j) \cap^{dnf} ctx_d(s_i)$, and thus will not be eliminated by the domination-based pruning.

A.3 Inter-task Pruning

It is straightforward to prove, from the definition of inter-task pruning, that it only eliminates invalid/dominated compositions from the search space.

Appendix B

Metadata Estimates for Hierarchy Concepts

In the following table, we provide the estimates adopted in our case study evaluation to fill in the missing metadata values for the candidate learning objects of each concept of the composition hierarchy.

Table B.1: Learning Time, Size, and Cost Estimates for Concepts

Concept	**Learning Time (minutes)**	**Size (number of pages)**	**Cost (£)**
0. Algorithms and Data Structures	Min: 565 Max: 1675	Min: 31 Max: 109	Min: 0 Max: 535
1. Basic Data Structures	Min: 120 Max: 430	Min: 8.5 Max: 31	Min: 0 Max: 120
1.1. Primitive Data Types	Min: 45 Max: 130	Min: 3.5 Max: 11	Min: 0 Max: 45
1.1.1. Boolean	Min: 5 Max: 10	Min: 0.5 Max: 1	Min: 0 Max: 5

Continued on next page

Table B.1 – continued from previous page

Concept	**Learning Time (minutes)**	**Size (number of pages)**	**Cost (£)**
1.1.2. Integer	Min: 5 Max: 10	Min: 0.5 Max: 1	Min: 0 Max: 5
1.1.3. Real	Min: 5 Max: 10	Min: 0.5 Max: 1	Min: 0 Max: 5
1.1.4. Char	Min: 5 Max: 10	Min: 0.5 Max: 1	Min: 0 Max: 5
1.1.5. Enumerated	Min: 10 Max: 30	Min: 0.5 Max: 3	Min: 0 Max: 10
1.1.6. Pointers	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 15
1.2. Composite Data Types	Min: 30 Max: 120	Min: 2 Max: 8	Min: 0 Max: 30
1.2.1. Arrays	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 15
1.2.2. Records	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 15
1.3. Abstract Data Types	Min: 45 Max: 180	Min: 3 Max: 12	Min: 0 Max: 45
1.3.1. List	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 15
1.3.2. Stack	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 15
1.3.3. Queue	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 15
2. Recursion	Min: 30 Max: 90	Min: 1 Max: 5	Min: 0 Max: 30
3. Sorting	Min: 150 Max: 300	Min: 5 Max: 15	Min: 0 Max: 100
3.1. Selection Sort	Min: 30 Max: 60	Min: 1 Max: 3	Min: 0 Max: 20

Continued on next page

Table B.1 – continued from previous page

Concept	**Learning Time (minutes)**	**Size (number of pages)**	**Cost (£)**
3.2. Insertion Sort	Min: 30 Max: 60	Min: 1 Max: 3	Min: 0 Max: 20
3.3. Bubble Sort	Min: 30 Max: 60	Min: 1 Max: 3	Min: 0 Max: 20
3.4. Quick Sort	Min: 30 Max: 60	Min: 1 Max: 3	Min: 0 Max: 20
3.5. Merge Sort	Min: 30 Max: 60	Min: 1 Max: 3	Min: 0 Max: 20
4. Trees	Min: 160 Max: 495	Min: 10.5 Max: 35	Min: 0 Max: 165
4.1. Binary Trees	Min: 75 Max: 210	Min: 5 Max: 15	Min: 0 Max: 70
4.1.1. Tree Representation	Min: 15 Max: 30	Min: 1 Max: 3	Min: 0 Max: 10
4.1.2. Tree Traversal	Min: 60 Max: 180	Min: 4 Max: 12	Min: 0 Max: 60
4.1.2.1. Preorder Traversal	Min: 15 Max: 45	Min: 1 Max: 3	Min: 0 Max: 15
4.1.2.2. Postorder Traversal	Min: 15 Max: 45	Min: 1 Max: 3	Min: 0 Max: 15
4.1.2.3. Inorder Traversal	Min: 15 Max: 45	Min: 1 Max: 3	Min: 0 Max: 15
4.1.2.4. Levelorder Traversal	Min: 15 Max: 45	Min: 1 Max: 3	Min: 0 Max: 15
4.2. Binary Search Trees 55	Min: 3.5 Max: 165	Min: Max: 12	Min: 0 Max: 55
4.2.1. Binary Search Trees' Properties	Min: 10 Max: 30	Min: 0.5 Max: 3	Min: 0 Max: 10

Continued on next page

Table B.1 – continued from previous page

Concept	**Learning Time (minutes)**	**Size (number of pages)**	**Cost (£)**
4.2.2. Insertion	Min: 15 Max: 45	Min: 1 Max: 3	Min: 0 Max: 15
4.2.2. Searching	Min: 15 Max: 45	Min: 1 Max: 3	Min: 0 Max: 15
4.2.2. Deletion	Min: 15 Max: 45	Min: 1 Max: 3	Min: 0 Max: 15
4.3. AVL Trees	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 20
4.4. B-Trees	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 20
5. Graphs	Min: 75 Max: 240	Min: 4 Max: 15	Min: 0 Max: 80
5.1. Graph Representation	Min: 30 Max: 60	Min: 1 Max: 4	Min: 0 Max: 20
5.1.1. Adjacency List	Min: 15 Max: 30	Min: 0.5 Max: 2	Min: 0 Max: 10
5.1.2. Adjacency Matrix	Min: 15 Max: 30	Min: 0.5 Max: 2	Min: 0 Max: 10
5.2. Graph Traversal	Min: 30 Max: 120	Min: 2 Max: 8	Min: 0 Max: 40
5.2.1. Depth-first Search	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 20
5.2.2. Breadth-first Search	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 20
5.3. Topological Sort	Min: 15 Max: 60	Min: 1 Max: 3	Min: 0 Max: 20
6. Backtracking	Min: 30 Max: 120	Min: 2 Max: 8	Min: 0 Max: 40
6.1. Backtracking Algorithms	Min: 15	Min: 1	Min: 0

Continued on next page

Table B.1 – continued from previous page

Concept	**Learning Time (minutes)**	**Size (number of pages)**	**Cost (£)**
	Max: 60	Max: 4	Max: 20
6.2. Optimal Selection Problem	Min: 15 Max: 60	Min: 1 Max: 4	Min: 0 Max: 20

References

[1] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, July 2003.

[2] R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in METEOR-S. In *Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 23–30, 2004.

[3] H. Akkermans, Z. Baida, J. Gordijn, N. Pena, A. Altuna, and I. Laresgoiti. Value webs: Using ontologies to bundle real-world services. *IEEE Intelligent Systems*, 19(4):57–66, 2004.

[4] G. Alonso, R. Gunthor, M. Kamath, D. Agrawal, A. E. Abbadi, and C. Mohan. Exotica/FMDC: A workflow management system for mobile and disconnected clients. In *Databases and Mobile Computing*, pages 27–45. 1996.

[5] M. Alrifai, D. Skoutas, and T. Risse. Selecting skyline services for QoS-based web service composition. In *Proc. of the 19th Int. Conf. on world wide web*, pages 11–20, 2010.

[6] M. Alrifai and R. Thomas. Combining global optimization with local selection for efficient QoS-aware service composition. In *Proceedings of the 18th international conference on World Wide Web*, pages 881–890, New York, NY, USA, 2009. ACM.

[7] M. Alt, A. Hoheisel, H. Pohl, and S. Gorlatch. A grid workflow language using high-level petri nets. In *Parallel Processing and Applied Mathematics*, volume 3911 of *Lecture Notes in Computer Science*, pages 715–722. 2006.

[8] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. GridAnt: a client-controllable grid workflow system. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004.

[9] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[10] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, September 2005.

[11] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web service description for the semantic web. In *The Semantic Web - ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pages 348–363. Springer Berlin Heidelberg, 2002.

[12] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. PAWS: A framework for executing adaptive web-service processes. *IEEE Software*, 24:39–46, 2007.

[13] D. Ardagna and B. Pernici. Global and local QoS constraints guarantee in web service selection. In *Proceedings of the 2005 IEEE International Conference on Web Services*, pages 805–806, 2005.

[14] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33:369–384, 2007.

[15] R. Aschoff and A. Zisman. QoS-driven proactive adaptation of service composition. In *Proceedings of the 9th international conference on Service-Oriented Computing*, pages 421–435, Berlin, Heidelberg, 2011. Springer-Verlag.

[16] R. R. Aschoff and A. Zisman. Proactive adaptation of service composition. In *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 1–10, 2012.

[17] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[18] Z. Azmeh, F. Hamoui, M. Huchard, N. Messai, C. Tibermacine, C. Urtado, and S. Vauttier. Backing composite web services using formal concept analysis. In *Formal Concept Analysis*, volume 6628 of *Lecture Notes in Computer Science*, pages 26–41. Springer Berlin Heidelberg, 2011.

[19] Z. Azmeh, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier. Using concept lattices to support web service compositions with backup services. In *Proceedings of the Fifth International Conference on Internet and Web Applications and Services*, pages 363–368, 2010.

[20] P. Balatsoukas, A. Morris, and A. O'Brien. Learning objects update: Review and critical approach to content aggregation. *Educational Technology and Society*, 11(2):119–130, 2008.

[21] L. Barakat, S. Miles, and M. Luck. Efficient correlation-aware service selection. In *Proceedings of the 2012 IEEE 19th International Conference on Web Services*, pages 1–8, 2012.

[22] L. Barakat, S. Miles, and M. Luck. Reactive service selection in dynamic service environments. In *Proceedings of the First European Conference on Service-Oriented and Cloud Computing*, pages 17–31, 2012.

[23] L. Barakat, S. Miles, I. Poernomo, and M. Luck. Efficient multi-granularity service composition. In *Proceedings of the 2011 IEEE 18th International Conference on Web Services*, pages 227–234, 2011.

[24] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. J. MDovey, D. Feygin, A. Hately, R. Kochman, P. Macias, M. Novotny, M. Paolucci, C. von Riegen, T. Rogers, K. Sycara, P. Wenzel, and Z. Wu. UDDI Version 3.0.2 http://uddi.org/pubs/uddi_v3.htm, 2004.

[25] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz. Dynamic replanning of web service workflows. In *Proceedings of the IEEE International Conference on Digital Ecosystems and Technologies*, pages 211–216, 2007.

[26] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World-Wide Web. *Commun. ACM*, 37(8):76–82, August 1994.

[27] S. Bhiri, O. Perrin, and C. Godart. Ensuring required failure atomicity of composite web services. In *Proceedings of the 14th international conference on World Wide Web*, pages 138–147. ACM, 2005.

[28] S. Bhiri, O. Perrin, and C. Godart. Extending workflow patterns with transactional dependencies to define reliable composite web services. In *Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, page 145, 2006.

[29] A. Bucchiarone, M. Pistore, H. Raik, and R. Kazhamiakin. Adaptation of service-based business processes by context-aware replanning. In *IEEE International Conference on Service-Oriented Computing and Applications*, pages 1–8, 2011.

[30] G. Canfora, M. D. Penta, R. Esposito, and M . L. Villani. QoS-Aware replanning of composite web services. In *Proceedings of the IEEE International Conference on Web Services*, pages 121–129. IEEE Computer Society, 2005.

[31] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1069–1075, 2005.

[32] V. Cardellini, E. Casalicchio, V. Grassi, F. L. Presti, and R. Mirandola. QoS-driven runtime adaptation of service oriented architectures. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 131–140, New York, NY, USA, 2009. ACM.

[33] J. Cardoso, J. Miller, A. Sheth, and J. Arnold. Quality of service for workflows and web service processes. *Web Semantics*, 1:281–308, 2004.

[34] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and dynamic service composition in eFlow. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 13–31, London, UK, 2000. Springer-Verlag.

[35] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in web service composition and execution. In *Proceedings of the 2006 IEEE International Conference on Web Services*, pages 549–557, 2006.

[36] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C note 15 March 2001, W3C, 2001.

[37] M. Christopher. The agile supply chain: competing in volatile markets. *Industrial Marketing Management*, 29(1):37–44, 2000.

[38] D. B. Claro, P. Albers, and J. K. Hao. Selecting web services for optimal composition. In *Proceedings of the 2nd International Workshop on Semantic and Dynamic Web Processes*, pages 32–45, 2005.

[39] E. Cohen and M. Nycz. Learning objects and e-learning: an informing science perspective. *Interdisciplinary Journal of E-Learning and Learning Objects*, 2(1):23–34, 2006.

[40] M. Colombo, E.D. Nitto, and M. Mauri. SCENE: A service composition execution environment supporting dynamic changes disciplined through rules. In *Proceedings of the 4th International Conference on Service-Oriented Computing*, Lecture Notes in Computer Science, pages 191–202. Springer, 2006.

[41] M. Comuzzi and B. Pernici. An architecture for flexible web service QoS negotiation. In *Ninth IEEE International EDOC Enterprise Computing Conference*, pages 70–79, 2005.

[42] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *Communications of the ACM*, 46(10):29–34, October 2003.

[43] F. Curbera, R. Khalaf, W. A. Nagy, and S. Weerawarana. Implementing BPEL4WS: the architecture of a BPEL4WS implementation. *Concurrency and Computation: Practice and Experience*, 18(10):1219–1228, August 2006.

[44] Y. Dai, L. Yang, and B. Zhang. QoS-driven self-healing web service composition based on performance prediction. *Computer Science and Technology*, 24:250–261, March 2009.

[45] A. D'Ambrogio. A model-driven WSDL extension for describing the QoS of web services. In *IEEE International Conference on Web Services*, pages 789–796, 2006.

[46] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Grid Computing*, 1(1):25–39, 2003.

[47] N. Delgado, A.Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *IEEE Transactions on Software Engineering*, 30(12):859–872, 2004.

[48] E. Duval and W. Hodgins. A LOM research agenda. In *Proceedings of the 2003 International Conference on World Wide Web*, pages 1–9, 2003.

[49] R. Farrell, S. D. Liburd, and J. C. Thomas. Dynamic assembly of learning objects. In *Proceedings of the 2004 International Conference on World Wide Web*, pages 162–169. ACM Press, 2004.

[50] F. Gagliardi, B. Jones, F. Grey, M. E. Bégin, and M. Heikkurinen. Building an infrastructure for scientific grid computing: status and goals of the EGEE project. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 363(1833):1729–1742, 2005.

[51] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, 2007.

[52] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to web services architecture. *IBM Systems*, 41(2):170–177, 2002.

[53] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. SOAP version 1.2 part 1: Messaging framework (second edition). W3C recommendation 27 April 2007, W3C, 2007.

[54] H. Guo, F. Tao, L. Zhang, S. Su, and N. Si. Correlation-aware web services composition and QoS computation model in virtual enterprise. *The International Journal of Advanced Manufacturing Technology*, 51(5-8):817–827, 2010.

[55] J. El Hadad, M. Manouvrier, and M. Rukoz. TQoS: Transactional and QoS-aware selection algorithm for automatic web service composition. *IEEE Transactions on Services Computing*, 3(1):73–85, 2010.

[56] R. Haesen, M. Snoeck, W. Lemahieu, and S. Poelmans. On the definition of service granularity and its architectural impact. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, pages 375–389, 2008.

[57] C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering*, 26(10):943–958, 2000.

[58] J. Hielscher, R. Kazhamiakin, A. Metzger, and M. Pistore. A framework for proactive self-adaptation of service-based applications based on online testing. In *Proceedings of the 1st European Conference on Towards a Service-Based Internet*, pages 122–133, 2008.

[59] N. Hiratsuka, F. Ishikawa, and S. Honiden. Service selection with combinational use of functionally-equivalent services. In *Proceedings of the 2011 IEEE International Conference on Web Services*, pages 97–104, 2011.

[60] A. Hoheisel. User tools and languages for graph-based grid workflows. *Concurrency and Computation: Practice and Experience*, 18(10):1101–1113, 2006.

[61] M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[62] Learning Technology Standards Committee IEEE. Draft standard for learning object metadata. Technical report, 2002.

[63] D. Ivanovic, M. Carro, and M. Hermenegildo. Towards data-aware QoS-driven adaptation for service orchestrations. In *Proceedings of the International Conference on Web Services*, pages 107–114, 2010.

[64] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl. QoS aggregation for web service composition using workflow patterns. In *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference*, pages 149–159, 2004.

[65] M.C. Jaeger and H. Ladner. Improving the QoS of WS compositions based on redundant services. In *Proceedings of the International Conference on Next Generation Web Services Practices*, pages 189–194, 2005.

[66] S. Kalasapur, M. Kumar, and B. Shirazi. Dynamic service composition in pervasive computing. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):907–918, 2007.

[67] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. Provenance trails in the Wings-Pegasus system. *Concurrency And Computation: Practice And Experience*, 20(5):587–597, 2008.

[68] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *Proceedings of the 5th international joint conference on Autonomous agents and multiagent systems*, pages 915–922, 2006.

[69] M. Klusch, A. Gerber, and M. Schmidt. Semantic web service composition planning with OWLS-XPlan. In *Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, pages 55–62, 2005.

[70] M. Klusch, P. Kapahnke, and I. Zinnikus. Hybrid adaptive web service selection with SAWSDL-MX and WSDL-Analyzer. In *The Semantic Web: Research and Applications*, volume 5554 of *Lecture Notes in Computer Science*, pages 550–564. Springer Berlin Heidelberg, 2009.

[71] M. Klusch and F. Kaufer. WSMO-MX: A hybrid semantic web service matchmaker. *Web Intelligence and Agent Systems*, 7(1):23–42, January 2009.

[72] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic annotations for WSDL and XML schema. *IEEE Internet Computing*, 11(6):60–67, 2007.

[73] K. Kritikos and D. Plexousakis. Semantic QoS metric matching. In *4th European Conference on Web Services*, pages 265–274, 2006.

[74] M. Lackovic, D. Talia, R. Tolosana-Calasanz, J. A. Banares, and O. F. Rana. A taxonomy for the analysis of scientific workflow faults. In *Proceedings of the 13th IEEE International Conference on Computational Science and Engineering*, pages 398–403, 2010.

[75] F. Lecue and N. Mehandjiev. Seeking quality of web service composition in a semantic dimension. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):942–959, 2011.

[76] L. Li, J. Wei, and T. Huang. High performance approach for Multi-QoS constrained web services selection. In *Proceedings of the 5th international conference on Service-Oriented Computing*, pages 283–294, Berlin, Heidelberg, 2007. Springer-Verlag.

[77] K. Lin, H. Lu, T. Yu, and C. Tai. A reputation and trust management broker framework for web applications. In *International Conference on e-Technology, e-Commerce, and e-Services*, pages 262–269. IEEE, 2005.

[78] K. J. Lin, J. Zhang, Y. Zhai, and B. Xu. The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA. *Service Oriented Computing and Applications*, 4(3):157–168, 2010.

[79] A. Liu, L. Huang, and Q. Li. QoS-aware web services composition using transactional composition operator. In *Proceedings of the 7th international conference on Advances in Web-Age Information Management.*

[80] Y. Liu, A. H. Ngu, and L. Z. Zeng. QoS computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, 2004.

[81] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic markup for web services. W3C member submission 22 November 2004, W3C, 2004.

[82] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing semantics to web services: The OWL-S approach. In *Semantic Web Services and Web Process Composition*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42. 2005.

[83] E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004.

[84] E. M. Maximilien and M. P. Singh. Toward autonomic web services trust and selection. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 212–221, New York, NY, USA, 2004. ACM.

[85] E. M. Maximilien and M. P. Singh. Multiagent system for dynamic web services selection. In *Proceedings of 1st Workshop on Service-Oriented Computing and Agent-Based Engineering*, pages 25–29, 2005.

[86] R. E. Mayer. Elements of a science of e-learning. *Educational Computing Research*, 29(3):297–313, 2003.

[87] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[88] D. A. Menascé, E. Casalicchio, and V. Dubey. A heuristic approach to optimal service selection in service oriented architectures. In *Proceedings of the 7th international workshop on Software and performance*, pages 13–24, 2008.

[89] A. Metzger, O. Sammodi, and K. Pohl. Accurate proactive adaptation of service-oriented systems. In *Assurances for Self-Adaptive Systems*, volume 7740 of *Lecture Notes in Computer Science*, pages 240–265. Springer Berlin Heidelberg, 2013.

[90] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka. Towards pro-active adaptation with confidence: augmenting service monitoring with online testing. In *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 20–28, 2010.

[91] T. Mikalsen, S. Tai, and I. Rouvellou. Transactional attitudes: Reliable composition of autonomous web services. In *Workshop on Dependable Middleware-based Systems WDMS 2002*, 2002.

[92] T. Moller and H. Schuldt. OSIRIS next: Flexible semantic failure handling for composite web service execution. In *IEEE International Conference on Semantic Computing*, pages 212–217, 2010.

[93] L. D. Ngan, M. Kirchberg, and R. Kanagasabai. Review of semantic web service discovery methods. In *6th World Congress on Services*, pages 176–177, 2010.

[94] E. D. Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15:313–341, 2008.

[95] X. Ochoa, J. Klerkx, B. Vandeputte, and E. Duval. On the use of learning object metadata: the GLOBE experience. In *Proceedings of the 6th European conference on Technology enhanced learning: towards ubiquitous learning*, pages 271–284, 2011.

[96] M. Palacios, J. Garcia-Fanjul, and J. Tuya. Testing in service oriented architectures with dynamic binding: A mapping study. *Information and Software Technology*, 53(3):171–189, 2011.

[97] M. P. Papazoglou, S. Benbernou, and V. Andrikopoulos. On the evolution of services. *IEEE Transactions on Software Engineering*, 38(3):609–628, 2012.

[98] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing. *Communications of the ACM*, 46:25–28, 2003.

[99] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[100] J. Patel, W. T. L. Teacy, N. R. Jennings, M. Luck, S. Chalmers, G. Shercliff, P. J. Stockreisser, J. Shao, W. A. Gray, N. J. Fiddian, and S. Thompson. Agent-based virtual organisations for the grid. *International Journal of Multi-Agent and Grid Systems*, 2005.

[101] L. Qi, Y. Tang, W. Dou, and J. Chen. Combining local optimization and enumeration for QoS-aware web service composition. In *Proceedings of the IEEE International Conference on Web Services*, pages 34–41, 2010.

[102] G. Raines. Cloud computing and SOA. Technical report, Service-oriented Architecture (SOA) Series, the MITRE Corporation, 2009.

[103] S. Ran. A model for web services discovery with QoS. *ACM SIGecom Exchanges*, 4(1):1–10, 2003.

[104] J. Rao and X. Su. A survey of automated web service composition methods. In *SWSWPC: Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition*, pages 43–54, 2004.

[105] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, January 2005.

[106] J. Sabater and C. Sierra. REGRET: A reputation model for gregarious societies. In *Fourth Workshop on Deception Fraud and Trust in Agent Societies*, pages 61–70, 2001.

[107] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(3):1–42, 2010.

[108] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl. Usage-based online testing for proactive adaptation of service-based applications. In *Proceedings of the IEEE 35th Annual Conference on Computer Software and Applications*, pages 582–587, 2011.

[109] R. Shahin, L. Barakat, S. Mahmoud, and M. Alkassar. Dynamic generation of adaptive courses. In *Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, pages 1–4, 2008.

[110] E. Sirin, B. Parsia, and J. Hendler. Template-based composition of semantic web services. In *Proceedings of the AAAI Fall Symposium on Agents and the Semantic Web*, pages 85—-92, 2005.

[111] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for web service composition using SHOP2. *Web Semantics*, 1:377–396, 2004.

[112] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:61–70, 1981.

[113] N. Srinivasan, M. Paolucci, and K K. Sycara. Semantic web service discovery in the OWL-S IDE. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, volume 6, 2006.

[114] S. Stein, T. R. Payne, and N. R. Jennings. Flexible provisioning of web service workflows. *ACM Transactions on Internet Technology*, 9(1):1–45, 2009.

[115] S. Stein, T. R. Payne, and N. R. Jennings. Robust execution of service workflows using redundancy and advance reservations. *IEEE Transactions on Services Computing*, 4(2):125–139, April 2011.

[116] S. A. Sutton and J. Mason. The dublin core and metadata for educational resources. In *Proceeding of the International Conference on Dublin Core and Metadata Applications*, pages 25–31, 2001.

[117] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):27–46, 2003.

[118] F. Tao, D. Zhao, H. Yefa, and Z. Zhou. Correlation-aware resource service composition and optimal-selection in manufacturing grid. *European Journal of Operational Research*, 201(1):129–143, 2010.

[119] W. T. L. Teacy, J. Patel, N. R. Jennings, and M. Luck. TRAVOS: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-Agent Systems*, 12(2):183–198, 2006.

[120] R. Tolosana-Calasanz, J. A. Bañares, P. Álvarez, J. Ezpeleta, and O. F. Rana. Exception handling patterns for hierarchical scientific workflows. In *Proceedings of the 6th international workshop on Middleware for grid computing*, pages 1–6, 2008.

[121] R. Tolosana-Calasanz, J. A. Bañares, O. F. Rana, P. Álvarez, J. Ezpeleta, and A. Hoheisel. Adaptive exception handling for scientific workflows. *Concurrency and Computation: Practice and Experience*, 22(5):617–642, 2010.

[122] I. Toma, D. Foxvog, and M. C. Jaeger. Modeling QoS characteristics in WSMO. In *Proceedings of the 1st workshop on Middleware for Service Oriented Computing*, pages 42–47, 2006.

[123] D. Tosi, G. Denaro, and M. Pezze. Towards autonomic service-oriented applications. *International Journal of Autonomic Computing*, 1(1):58–80, 2009.

[124] V. Tosic, D. Mennie, and B. Pagurek. On dynamic service composition and its applicability to e-business software systems - the ICARIS experience. In *Workshop on on Object-Oriented Business Systemsat the 15th European Conference on Object-Oriented Programming*, pages 95–108, 2000.

[125] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *The Semantic Web*, volume 3298 of *Lecture Notes in Computer Science*, pages 380–394. 2004.

[126] M. Treiber, H. Truong, and S. Dustdar. On analyzing evolutionary changes of web services. In *Proceedings of the International Conference on Service-Oriented Computing Workshops*, pages 284–297, 2009.

[127] I. Trummer and B. Faltings. Optimizing the tradeoff between discovery, composition, and execution cost in service composition. In *Proceedings of the IEEE International Conference on Web Services*, pages 476–483, 2011.

[128] C. Summerfield L. Zealey V. Avery, E. Chamberlain. *Focus on the Digital Age*. Focus On. Palgrave Macmillan, 2007.

[129] W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[130] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[131] K. Verma, R. Akkiraju, R. Goodwin, P. Doshi, and J. Lee. On accommodating inter service dependencies in web process flow composition. In *AAAI Spring Symposium on SWS*, pages 37–43, 2004.

[132] L. Vu, M. Hauswirth, and K. Aberer. QoS-based service selection and ranking with trust and reputation management. In *Proceedings of the Cooperative Information System Conference*, pages 446–483, 2005.

[133] F. Wagner, B. Kloepper, F. Ishikawa, and S. Honiden. Towards robust service compositions in the context of functionally diverse services. In *Proceedings of the 21st international conference on World Wide Web*, pages 969–978, 2012.

[134] H. H. Wang, N. Gibbins, T. R. Payne, and D. Redavid. A formal model of the semantic web service ontology (WSMO). *Information Systems*, 37(1):33–60, 2012.

[135] X. Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the 12th International Conference on Machine Learning*, pages 549–557, 1995.

[136] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A qos-aware selection model for semantic web services. In *Proceedings of the 4th international conference on Service-Oriented Computing*, ICSOC'06, pages 390–401, 2006.

[137] M. Weske and G. Vossen. Workflow languages. In *Handbook on Architectures of Information Systems*, pages 359–379. Springer Berlin Heidelberg, 1998.

[138] Q. Wu, Q. Zhu, and M. Zhou. A correlation-driven optimal service selection approach for virtual enterprise establishment. *Intelligent Manufacturing*, pages 1–13, 2013.

[139] L. Yang, Y. Dai, and B. Zhang. Performance prediction based EX-QoS driven approach for adaptive service composition. *Information Science and Engineering*, 25(2):345–362, 2009.

[140] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. Technical report, Grid Computing, 2005.

[141] T. Yu and K. Lin. Adaptive algorithms for finding replacement services in autonomic distributed business processes. In *Proceedings of the International Symposium on Autonomous Decentralized Systems*, pages 427–434, 2005.

[142] T. Yu and K. Lin. A broker-based framework for QoS-aware web service composition. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 22–29, 2005.

[143] T. Yu, Y. Zhang, and K. Lin. Efficient algorithms for web services selection with end-to-end QoS constraints. *ACM Transactions on the Web*, 1(1), 2007.

[144] X. Yuan and X.Liu. Heuristic algorithms for multi-constrained quality of service routing. *IEEE/ACM Transactions on Networking*, 10(2):244–256, 2002.

[145] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th International Conference on World Wide Web*, pages 411–421, 2003.

[146] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.

[147] D. Zhang, M. Chen, and L. Zhou. Dynamic and personalized web services composition in e-business. *Information Systems Management*, 22(3):50–65, 2005.

[148] M. Zhang, C. Liu, J. Yu, Z. Zhu, and B. Zhang. A correlation context-aware approach for composite service selection. *Concurrency and Computation: Practice and Experience*, 25(13):1909–1927, 2013.

[149] A. Zisman, J. Dooley, and G. Spanoudakis. Proactive runtime service discovery. In *Proceedings of the IEEE International Conference on Services Computing*, volume 1, pages 237–245, 2008.